PyNiryo2 Documentation

This documentation presents Ned's PyPi package, which is the second version of the python API Pyniryo for the robots: Niryo One, Ned and Ned2. It is based on the roslibpy library and allows a more complete programming of the robots.

It offers a simple way for developers to create programs for robot and to control them via remote communication from their computers. Contrary to the Python ROS Wrapper, the user will not need to be connected on the robot through a terminal. This API is based on roslibpy (https://roslibpy.readthedocs.io/en/latest/index.html).

Note

This package is able to control Ned in simulation as well as the physical robot.

Niryo Ned

Functionality	PyNiryo	PyNiryo2
Robot control	**YES**	**YES**
Callbacks functions	**NO**	**YES**
Parallelism of commands	**NO**	**YES**
Asynchronous functions	**NO**	**YES**

Before getting started

If you haven't already done so , make sure to learn about the ROS robot software by reading theNed's Software documentation.

This documentation also contains everything you need to know if you want to use Ned through simulation.

Sections organization

This document is organized in 4 main sections

Setup

Install & Setup your environment in order to use Ned with PyNiryo2.

Firstly, follow Installation instructions (index.html#document-source/setup/installation), then find your Robot IP address (index.html#document-source/setup/ip_address) to be ready.

Installation

The library uses Python, which must be installed and available in your working environment

The version should be **equal or above** :

- 2.7 if you are using Python 2
- 3.6 if you are using Python 3

Hint

To check your Python version, use the command `python --version` if you are using Python 2 and `python3 --version` if you are using Python3

The below sections explain how to install the library with **pip**, the package installer for Python

Attention

If you have both Python 2 & Python 3 installed on your computer, the command `pip` will install packages in Python 2 version. You should use `pip3` instead in order to to target Python 3

Installation with pip

You need to have installed Numpy package beforehand :

```
pip install numpy
```

To install Ned's Python package via `pip` , simply execute:

```
pip install pyniryo2
```

You can find more information about the PyPi packagehere (https://pypi.org/project/pyniryo2/)

Uninstall

To uninstall the library use

```
pip uninstall pyniryo2
```

Find your Robot's IP address

In order to use your robot through TCP connection, you will firstly need to connect to it, which imply that you know its IP Address

The next sections explain how to find your robot IP according to your configuration:

- Hotspot mode
- Simulation or directly on the robot
- Direct Ethernet connection
- Computer and Robot Connected on the same router

Hotspot mode

If you are directly connected to your robot through its wifi, the IP Address you will need to use is `10.10.10.10`

Simulation or directly on the robot

In this situation, the Robot is running on the same computer as the client, the IP Address will be the localhost address `127.0.0.1`

Direct Ethernet connection

If you are directly connected to your robot with an ethernet cable, the static IP of your robot will be `169.254.200.200`

The reader should note that he may has to change his wired settings to allow the connection. See howConnect to Ned via Ethernet on Ubuntu (https://niryo.com/docs/niryo-one/developer-tutorials/connect-to-niryo-one-via-ethernet-on-ubuntu/)

Computer and Robot Connected on the same router

You will need to find the robot address using `nmap` , or you can also use search button of Niryo Studio to see which robots are available

You can also make IP permanent (https://docs.niryo.com/product/niryo-studio/source/settings.html#network-settings) so that you won't have to search for it next time.

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Verify your Setup and Get Started

In order to verify your computer's setup, we are going to run a program from it, and see if the robot answers as expected.

Note

Before verifying your setup, be sure that your physical robot (or simulation) is turned on

Firstly, go in the folder of your choice and create an empty file named "pyniryo_test.py". This file will contain the checking code

Edit this file and fill it with the following code

```
from pyniryo2 import *

robot_ip_address = "10.10.10.10"

# Connect to robot & calibrate
robot = NiryoRobot(robot_ip_address)
robot.arm.calibrate_auto()
# Move joints
robot.arm.move_joints([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
# Turn learning mode ON
robot.arm.set_learning_mode(True)
# Stop TCP connection
robot.end()
```

Attention

Replace the third line with your Robot IP Address (index.html#document-source/setup/ip_address) if you are not using Hotspot Mode

Still on your computer, open a terminal, and place your current directory in the same folder than your file. Then, run the command :

```
python pyniryo_test.py
```

Note

If you are using Python 3, you may need to change `python` to `python3`

If you robot start calibrating, then move, and finally, go to learning mode, your setup is validated, you can now start coding !

Examples

Learn how to use the PyNiryo2 package to implement various tasks

Examples: Basics

In this file, two short programs are implemented & commented in order to help you understand the philosophy behind the PyNiryo package

Danger

If you are using the real robot, make sure the environment around it is clear

Your first move joint

The following example shows a first use case. It's a simple MoveJ.

```
from pyniryo2 import *

robot = NiryoRobot("10.10.10.10")

robot.arm.calibrate_auto()

robot.arm.move_joints([0.2, -0.3, 0.1, 0.0, 0.5, -0.8])

robot.end()
```

Code Details - First Move J

First of all, we import the library to be able to access functions

```
from pyniryo2 import *
```

Then, we instantiate the connection and link the variable `robot` to the robot at the IP Address `10.10.10.10`

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
robot = NiryoRobot("10.10.10.10")
```

Once the connection is done, we calibrate the robot using its **calibrate_auto()** (index.html#pyniryo2.arm.arm.Arm.calibrate_auto) function

```
robot.arm.calibrate_auto()
```

As the robot is now calibrated, we can do a Move Joints by giving the 6 axis positions in radians ! To do so, we use **move_joints()** (index.html#pyniryo2.arm.arm.Arm.move_joints)

```
robot.arm.move_joints([0.2, -0.3, 0.1, 0.0, 0.5, -0.8])
```

Our process is now over, we can close the connection with **end()** (index.html#pyniryo2.niryo_robot.NiryoRobot.end)

```
robot.end()
```

Your first pick and place

In the second example, we are going to develop a pick and place algorithm

```
from pyniryo2 import *

robot = NiryoRobot("10.10.10.10")

robot.arm.calibrate_auto()
robot.tool.update_tool()

robot.tool.release_with_tool()
robot.arm.move_pose([0.2, -0.1, 0.25, 0.0, 1.57, 0.0])
robot.tool.grasp_with_tool()

robot.arm.move_pose([0.2, 0.1, 0.25, 0.0, 1.57, 0.0])
robot.tool.release_with_tool()

robot.end()
```

Code Details - First Pick And Place

First of all, we import the library and start the connection between our computer and the robot. We also calibrate the robot

```
from pyniryo2 import *

robot = NiryoRobot("10.10.10.10")
robot.arm.calibrate_auto()
```

Then, we equip the tool with **update_tool()** (index.html#pyniryo2.tool.tool.Tool.update_tool)

```
robot.tool.update_tool()
```

Now that our initialization is done, we can open the gripper (or push air from the vacuum) with **release_with_tool()** (index.html#pyniryo2.tool.tool.Tool.release_with_tool), go to the picking pose via **move_pose()** (index.html#pyniryo2.arm.arm.Arm.move_pose) & then catch an object with **grasp_with_tool()** (index.html#pyniryo2.tool.tool.Tool.grasp_with_tool) !

```
robot.tool.release_with_tool()
robot.arm.move_pose([0.2, -0.1, 0.25, 0.0, 1.57, 0.0])
robot.tool.grasp_with_tool()
```

We now get to the place pose, and place the object

```
robot.arm.move_pose([0.2, 0.1, 0.25, 0.0, 1.57, 0.0])
robot.tool.release_with_tool()
```

Our process is now over, we can close the connection

```
robot.end()
```

Notes

You may not have fully understood how to move the robot and use PyNiryo and that is totally fine because you will find more details on next examples page !

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

The important thing to remember from this page is how to import the library, connect to the robot & call functions

Examples: Movement

This document shows how to control Ned in order to make Move Joints & Move Pose.

If you want see more, you can look at PyNiryo - Arm (index.html#arm)

Important

In following sections, it is supposed that you are already connected to a calibrated robot. The robot instance is save in the variable `robot` . To know how to do so, go look at section Examples: Basics (index.html#document-source/examples/examples_basics)

Danger

If you are using the real robot, make sure the environment around it is clear

Joints

Move Joints

To make a moveJ, you need to pass :

- a list of 6 floats : `[j1, j2, j3, j4, j5, j6]`

It is possible to pass these parameters the function **move_joints()** (index.html#pyniryo2.arm.arm.Arm.move_joints) or a via the `joints` setter, at your convenience:

```
# Moving Joints with function & a list of floats
robot.arm.move_joints([-0.5, -0.6, 0.0, 0.3, 0.0, 0.0])

# Moving Joints with setter & 6 floats
robot.arm.joints = 0.2, -0.4, 0.0, 0.0, 0.0, 0.0

# Moving Joints with setter & a list of floats
robot.arm.joints = [-0.2, 0.3, 0.2, 0.3, -0.6, 0.0]
```

You should note that these 4 commands are doing exactly the same thing ! In your future scripts, chose the one you prefer, but try to remain consistent to keep a good readability

Get Joints

To get actual joint positions, you can use the function **get_joints()** (index.html#pyniryo2.arm.arm.Arm.get_joints) or the `joints` getter. Both will return a list of the 6 joints position:

```
# Getting Joints with function
joints_read = robot.arm.get_joints()

# Getting Joints with getter
joints_read = robot.arm.joints
```

Hint

As we are developing in Python, we can unpack list very easily, which means that we can retrieve joints value in 6 variables by writing `j1, j2, j3, j4, j5, j6 = robot.arm.get_joints()`

Pose

Move Pose

To perform a moveP, you can pass :

- a list of 6 floats : [x, y, z, roll, pitch, yaw]
- a **PoseObject** (index.html#pyniryo2.objects.PoseObject)

As for MoveJ, it is possible to pass these parameters the function **move_pose()** (index.html#pyniryo2.arm.arm.Arm.move_pose) or the `pose` setter, at your convenience:

```
pose_target = [0.2, 0.0, 0.2, 0.0, 0.0, 0.0]
pose_target_obj = PoseObject(0.2, 0.0, 0.2, 0.0, 0.0, 0.0)

# Moving Pose with function
robot.arm.move_pose(0.2, 0.0, 0.2, 0.0, 0.0, 0.0)
robot.arm.move_pose(pose_target)
robot.arm.move_pose(pose_target_obj)

# Moving Pose with setter
robot.arm.pose = (0.2, 0.0, 0.2, 0.0, 0.0, 0.0)
robot.arm.pose = pose_target
robot.arm.pose = pose_target_obj
```

Each of these 6 commands are doing the same thing

Get Pose

To get end effector actual pose, you can use the function **get_pose** (index.html#pyniryo2.arm.arm.Arm.get_pose) or the `pose` getter. Both will return a **PoseObject** (index.html#pyniryo2.objects.PoseObject)

```
# Getting Joints with function
pose_read = robot.arm.get_pose()

# Getting Joints with getter
pose_read = robot.arm.pose
```

How to use the PoseObject

The **PoseObject** (index.html#pyniryo2.objects.PoseObject) is a Python Object which allow to store all pose's 6 coordinates (x, y, z, roll, pitch, yaw) in one single instance. It can be converted into a list if needed with the method **to_list()**

It also allows to create new **PoseObject** (index.html#pyniryo2.objects.PoseObject) with some offset, much easier than copying list and editing only 1 or 2 values. For instance, imagine that we want to shift the place pose by 5 centimeters at each iteration of a for loop, you can use the **copy_with_offsets()** method:

```
pick_pose = PoseObject(
x=0.30, y=0.0, z=0.15,
roll=0, pitch=1.57, yaw=0.0
)
first_place_pose = PoseObject(
    x=0.0, y=0.2, z=0.15,
    roll=0, pitch=1.57, yaw=0.0
)
for i in range(5):
    robot.arm.move_pose(pick_pose)
    new_place_pose = first_place_pose.copy_with_offsets(x_offset=0.05 * i)
    robot.arm.move_pose(new_place_pose)
```

Examples: Tool Action

This page shows how to control Ned's tools

If you want see more, you can look at PyNiryo - Tools (index.html#tool)

Important

In this section, it is supposed that you are already connected to a calibrated robot. The robot instance is save in the variable `robot`

Danger

If you are using the real robot, make sure the environment around it is clear

Tool control

Equip Tool

In order to use a tool, it should be plugged mechanically to the robot but also connected software wise.

To do that, we should use the function **update_tool()** (index.html#pyniryo2.tool.tool.Tool.update_tool) which take no argument. It will scan motor connections and set the new tool !

The line to equip a new tool is

```
robot.tool.update_tool()
```

Note

For the Grasping and Releasing sections, this command should be added in your codes ! If you wan to use a specific tool, you need to store the **ToolID** you are using in a variable named tool_used

Grasping

To grasp with any tool, you can use the function **grasp_with_tool()** (index.html#pyniryo2.tool.tool.Tool.grasp_with_tool). This action correspond to :

- Close gripper for Grippers
- Pull Air for Vacuum pump
- Activate for Electromagnet

The line to grasp is

```
robot.tool.grasp_with_tool()
```

To grasp an object by specifying the tool

```
if tool_used in [ToolID.GRIPPER_1, ToolID.GRIPPER_2, ToolID.GRIPPER_3]:
    robot.tool.close_gripper(speed=500)
elif tool_used == ToolID.ELECTROMAGNET_1:
    pin_electromagnet = PinID.XXX
    robot.tool.setup_electromagnet(pin_electromagnet)
    robot.tool.activate_electromagnet(pin_electromagnet)
elif tool_used == ToolID.VACUUM_PUMP_1:
    robot.tool.pull_air_vacuum_pump()
```

Releasing

To release with any tool, you can use the function **release_with_tool()** (index.html#pyniryo2.tool.tool.Tool.release_with_tool). This action correspond to:

- Open gripper for Grippers
- Push Air for Vacuum pump
- Deactivate for Electromagnet

To release an object by specifying parameters

```
if tool_used in [ToolID.GRIPPER_1, ToolID.GRIPPER_2, ToolID.GRIPPER_3]:
    robot.tool.open_gripper(speed=500)
elif tool_used == ToolID.ELECTROMAGNET_1:
    pin_electromagnet = PinID.XXX
    robot.tool.setup_electromagnet(pin_electromagnet)
    robot.tool.deactivate_electromagnet(pin_electromagnet)
elif tool_used == ToolID.VACUUM_PUMP_1:
    robot.tool.push_air_vacuum_pump()
```

Pick & Place with tools

A Pick & Place is a action which consists in going to a certain pose in order to pick an object and then, going to another pose to place it.

This operation can be proceed as follows :

1. Going over the object with a certain offset to avoid collision
2. Going down until object's height
3. Grasping with tool
4. Going back to step 1's pose.
5. Going over the place pose with a certain offset to avoid collision
6. Going down until place's height
7. Releasing the object with tool
8. Going back to step 5's pose.

There is a plenty of ways to perform a pick and place with PyNiryo. Methods will be presented from the lowest to highest level

Code Baseline

For the sake of brevity, every piece of code beside the Pick & Place function won't be rewritten for every method. So that, you will need to use the code and implement the Pick & Place function to it

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```python
# Imports
from pyniryo2 import *

tool_used = ToolID.XXX  # Tool used for picking
robot_ip_address = "x.x.x.x" # Robot address

# The pick pose
pick_pose = PoseObject(
    x=0.25, y=0., z=0.15,
    roll=-0.0, pitch=1.57, yaw=0.0,
)
# The Place pose
place_pose = PoseObject(
    x=0.0, y=-0.25, z=0.1,
    roll=0.0, pitch=1.57, yaw=-1.57)

def pick_n_place_version_x(robot):
    # -- -------------- -- #
    # -- CODE GOES HERE -- #
    # -- -------------- -- #

if __name__ == '__main__':
    # Connect to robot
    client = NiryoRobot(robot_ip_address)
    # Calibrate robot if robot needs calibration
    client.arm.calibrate_auto()
    # Changing tool
    client.arm.update_tool()

    pick_n_place_version_x(client)

    # Releasing connection
    client.end()
```

First Solution : the heaviest

For this first function, everything steps is done by hand, as well as poses computing

Note

In this example, the tool used is a Gripper. If you want to use another tool than a gripper, do not forget to adapt grasp & release functions !

```python
def pick_n_place_version_1(robot):
    height_offset = 0.05  # Offset according to Z-Axis to go over pick & place poses
    gripper_speed = 400

    # Going Over Object
    robot.arm.move_pose(pick_pose.x, pick_pose.y, pick_pose.z + height_offset,
                        pick_pose.roll, pick_pose.pitch, pick_pose.yaw)
    # Opening Gripper
    robot.tool.open_gripper(gripper_speed)
    # Going to picking place and closing gripper
    robot.arm.move_pose(pick_pose)
    robot.tool.close_gripper(gripper_speed)

    # Raising
    robot.arm.move_pose(pick_pose.x, pick_pose.y, pick_pose.z + height_offset,
                        pick_pose.roll, pick_pose.pitch, pick_pose.yaw)

    # Going Over Place pose
    robot.arm.move_pose(place_pose.x, place_pose.y, place_pose.z + height_offset,
                        place_pose.roll, place_pose.pitch, place_pose.yaw)
    # Going to Place pose
    robot.arm.move_pose(place_pose)
    # Opening Gripper
    robot.tool.open_gripper(gripper_speed)
    # Raising
    robot.arm.move_pose(place_pose.x, place_pose.y, place_pose.z + height_offset,
                        place_pose.roll, place_pose.pitch, place_pose.yaw)
```

Second Solution : Use of PoseObject

For the second solution, we use a **PoseObject** (index.html#pyniryo2.objects.PoseObject) in order to calculate approach poses more easily

Note

To see more about **PoseObject** (index.html#pyniryo2.objects.PoseObject), go look at PoseObject dedicated section (index.html#how-to-use-the-poseobject)

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
def pick_n_place_version_2(robot):
    height_offset = 0.05  # Offset according to Z-Axis to go over pick & place poses

    pick_pose_high = pick_pose.copy_with_offsets(z_offset=height_offset)
    place_pose_high = place_pose.copy_with_offsets(z_offset=height_offset)

    # Going Over Object
    robot.arm.move_pose(pick_pose_high)
    # Opening Gripper
    robot.tool.release_with_tool()
    # Going to picking place and closing gripper
    robot.arm.move_pose(pick_pose)
    robot.tool.grasp_with_tool()
    # Raising
    robot.arm.move_pose(pick_pose_high)

    # Going Over Place pose
    robot.arm.move_pose(place_pose_high)
    # Going to Place pose
    robot.arm.move_pose(place_pose)
    # Opening Gripper
    robot.tool.release_with_tool(gripper_speed)
    # Raising
    robot.arm.move_pose(place_pose_high)
```

Third Solution : Pick from pose & Place from pose functions

For those who already look at the API Documentation, you may have see pick & place dedicated functions !

In this example, we use **pick_from_pose()** (index.html#pyniryo2.pick_place.pick_place.PickPlace.pick_from_pose) and **place_from_pose()** (index.html#pyniryo2.pick_place.pick_place.PickPlace.place_from_pose) in order to split our function in only 2 commands !

```
def pick_n_place_version_3(robot):
    # Pick
    robot.pick_place.pick_from_pose(pick_pose)
    # Place
    robot.pick_place.place_from_pose(place_pose)
```

Fourth Solution : All in one

The example exposed in the previous section could be useful if you want to do an action between the pick & the place phases.

For those who want to do everything in one command, you can use the **pick_and_place()** (index.html#pyniryo2.pick_place.pick_place.PickPlace.pick_and_place) function !

```
def pick_n_place_version_4(robot):
    # Pick & Place
    robot.pick_place.pick_and_place(pick_pose, place_pose, dist_smoothing=0.01)
```

Examples: Conveyor

This document shows how to use Ned's conveyor belt

If you want see more about Ned's conveyor belt functions, you can look at PyNiryo - Conveyor (index.html#conveyor)

Danger

If you are using the real robot, make sure the environment around it is clear

Simple Conveyor control

This short example show how to connect a conveyor and launch its motor (control it by setting its speed and direction) :

```
from pyniryo2 import *

# Connecting to robot
robot = NiryoRobot(<robot_ip_address>)

# Activating connexion with conveyor
conveyor_id = robot.conveyor.set_conveyor()

# Running conveyor at 50% of its maximum speed, in Forward direction
robot.conveyor.run_conveyor(conveyor_id, speed=50, direction=ConveyorDirection.FORWARD)

# Waiting 3 seconds
robot.wait(3)

# Stopping robot motor
robot.conveyor.stop_conveyor(conveyor_id)

# Deactivating connexion with conveyor
robot.conveyor.unset_conveyor(conveyor_id)
```

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Advanced Conveyor control

This example shows how to do a certain amount of pick & place by using the conveyor with the infrared sensor

```
from pyniryo2 import *

# -- Setting variables
sensor_pin_id = PinID.GPIO_1A

catch_nb = 5

# The pick pose
pick_pose = PoseObject(
    x=0.25, y=0., z=0.15,
    roll=-0., pitch=1.57, yaw=0.0,
)
# The Place pose
place_pose = PoseObject(
    x=0., y=-0.25, z=0.1,
    roll=0., pitch=1.57, yaw=-1.57)

# -- MAIN PROGRAM

# Connecting to robot
robot = NiryoRobot(<robot_ip_address>)

# Activating connexion with conveyor
conveyor_id = robot.conveyor.set_conveyor()

for i in range(catch_nb):
    robot.conveyor.run_conveyor(conveyor_id)
    while robot.io.digital_read(sensor_pin_id) == PinState.LOW:
        robot.wait(0.1)

    # Stopping robot motor
    robot.conveyor.stop_conveyor(conveyor_id)
    # Making a pick & place
    robot.pick_place.pick_and_place(pick_pose, place_pose)

# Deactivating connexion with conveyor
robot.conveyor.unset_conveyor(conveyor_id)
```

Examples: Vision

This page shows how to use Ned's vision set

If you want see more about Ned's vision functions, you can look at PyNiryo - Vision
If you want to see how to do image processing, go check out the Pyniryo (first version) doc.

Note

Even if you do not own a Vision Set, you can still realize these examples with the Gazebo simulation version

Danger

If you are using the real robot, make sure the environment around it is clear

Needed piece of code

Important

In order to achieve following examples, you need to have create a vision workspace. In this page, the workspace used is named `workspace_1` . To create it, the user should go on Niryo Studio !

As the examples start always the same, add the following lines at the beginning of codes

```python
# Imports
from pyniryo import *

# - Constants
workspace_name = "workspace_1"  # Robot's Workspace Name
robot_ip_address = "x.x.x.x"

# The pose from where the image processing happens
observation_pose = PoseObject(
    x=0.16, y=0.0, z=0.35,
    roll=0.0, pitch=1.57, yaw=0.0,
)
# Place pose
place_pose = PoseObject(
    x=0.0, y=-0.2, z=0.12,
    roll=0.0, pitch=1.57, yaw=-1.57
)

# - Initialization

# Connect to robot
robot = NiryoRobot(robot_ip_address)
# Calibrate robot if robot needs calibration
robot.arm.calibrate_auto()
# Updating tool
robot.tool.update_tool()

# --- -------------- --- #
# --- CODE GOES HERE --- #
# --- -------------- --- #

robot.end()
```

Hint

All the following examples are only of part of what can be made with the API in terms of vision. We advise you to look atAPI - Vision (index.html#vision) to understand more deeply

Simple Vision Pick & Place

The goal of a Vision Pick & Place is the same as a classical Pick & Place, with a close difference : the camera detects where the robot has to go in order to pick !

This short example show how to do your first vision pick using the **vision_pick()** (index.html#pyniryo2.vision.vision.Vision.vision_pick) function :

```python
robot.move_pose(observation_pose)
# Trying to pick target using camera
obj_found, shape_ret, color_ret = robot.vision.vision_pick(workspace_name)
if obj_found:
    robot.pick_place.place_from_pose(place_pose)

robot.arm.set_learning_mode(True)
```

Code Details - Simple Vision Pick and Place

To execute a Vision pick, we firstly need to go to a place where the robot will be able to see the workspace

```python
robot.arm.move_pose(observation_pose)
```

Then, we try to perform a vision pick in the workspace with the **vision_pick()** (index.html#pyniryo2.vision.vision.Vision.vision_pick) function

```python
obj_found, shape_ret, color_ret = robot.vision.vision_pick(workspace_name)
```

Variables `shape_ret` and `color_ret` are respectively of type **ObjectShape** and **ObjectColor** , and store the shape and the color of the detected object ! We won't use them for this first example.

The `obj_found` variable is a boolean which indicates whereas an object has been found and picked, or not. Thus, if the pick worked, we can go place the object at the place pose.

```python
if obj_found:
    robot.pick_place.place_from_pose(place_pose)
```

Finally, we turn learning mode on:

```python
robot.arm.set_learning_mode(True)
```

Note

If you obj_found variable indicates False , check that :

- Nothing obstruct the camera field of view
- Workspace's 4 markers are visible
- At least 1 object is placed fully inside the workspace

First conditioning via Vision

In most of use cases, the robot will need to perform more than one Pick & Place. In this example, we will see how to condition multiple objects according to a straight line

```
# Initializing variables
offset_size = 0.05
max_catch_count = 4

# Loop until enough objects have been caught
catch_count = 0
while catch_count < max_catch_count:
    # Moving to observation pose
    robot.arm.move_pose(observation_pose)

    # Trying to get object via Vision Pick
    obj_found, shape, color = robot.vision.vision_pick(workspace_name)
    if not obj_found:
        robot.wait(0.1)
        continue

    # Calculate place pose and going to place the object
    next_place_pose = place_pose.copy_with_offsets(x_offset=catch_count * offset_size)
    robot.pick_place.place_from_pose(next_place_pose)

    catch_count += 1

robot.arm.go_to_sleep()
```

Code Details - First Conditioning via Vision

We want to catch max_catch_count objects, and space each of them by offset_size meter

```
offset_size = 0.05
max_catch_count = 4
```

We start a loop until the robot has caught max_catch_count objects

```
catch_count = 0
while catch_count < max_catch_count:
```

For each iteration, we firstly go to the observation pose and then, try to make a vision pick in the workspace

```
robot.arm.move_pose(observation_pose)

obj_found, shape, color = robot.vision.vision_pick(workspace_name)
```

If the vision pick failed, we wait 0.1 second and then, start a new iteration

```
if not obj_found:
    robot.wait(0.1)
    continue
```

Else, we compute the new place position according to the number of catches, and then, go placing the object at that place

```
next_place_pose = place_pose.copy_with_offsets(x_offset=catch_count * offset_size)
robot.pick_place.place_from_pose(next_place_pose)
```

We also increment the catch_count variable

```
catch_count += 1
```

Once the target catch number is achieved, we go to sleep

```
robot.arm.go_to_sleep()
```

Multi Reference Conditioning

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

During a conditioning task, objects may not always be placed as the same place according to their type. In this example, we will see how to align object according to their color, using the color element **ObjectColor** returned by **vision_pick()** (index.html#pyniryo2.vision.vision.Vision.vision_pick) function

```
# Distance between elements
offset_size = 0.05
max_failure_count = 3

# Dict to write catch history
count_dict = {
    ObjectColor.BLUE: 0,
    ObjectColor.RED: 0,
    ObjectColor.GREEN: 0,
}

try_without_success = 0
# Loop until too much failures
while try_without_success < max_failure_count:
    # Moving to observation pose
    robot.arm.move_pose(observation_pose)
    # Trying to get object via Vision Pick
    obj_found, shape, color = robot.vision.vision_pick(workspace_name)
    if not obj_found:
        try_without_success += 1
        robot.wait(0.1)
        continue

    # Choose X position according to how the color line is filled
    offset_x_ind = count_dict[color]

    # Choose Y position according to ObjectColor
    if color == ObjectColor.BLUE:
        offset_y_ind = -1
    elif color == ObjectColor.RED:
        offset_y_ind = 0
    else:
        offset_y_ind = 1

    # Going to place the object
    next_place_pose = place_pose.copy_with_offsets(x_offset=offset_x_ind * offset_size,
                                                   y_offset=offset_y_ind * offset_size)
    robot.pick_place.place_from_pose(next_place_pose)

    # Increment count
    count_dict[color] += 1
    try_without_success = 0

robot.arm.go_to_sleep()
```

Code Details - Multi Reference Conditioning

We want to catch objects until Vision Pick failed `max_failure_count` times. Each of the object will be put on a specific column according to its color. The number of catches for each color will be store on a dictionary `count_dict`

```
# Distance between elements
offset_size = 0.05
max_failure_count = 3

# Dict to write catch history
count_dict = {
    ObjectColor.BLUE: 0,
    ObjectColor.RED: 0,
    ObjectColor.GREEN: 0,
}

try_without_success = 0
# Loop until too much failures
while try_without_success < max_failure_count:
```

For each iteration, we firstly go to the observation pose and then, try to make a vision pick in the workspace

```
robot.move_pose(observation_pose)

obj_found, shape, color = robot.vision.vision_pick(workspace_name)
```

If the vision pick failed, we wait 0.1 second and then, start a new iteration, without forgetting the increment the failure counter

```
if not obj_found:
    try_without_success += 1
    robot.wait(0.1)
    continue
```

Else, we compute the new place position according to the number of catches, and then, go placing the object at that place

```
# Choose X position according to how the color line is filled
offset_x_ind = count_dict[color]

# Choose Y position according to ObjectColor
if color == ObjectColor.BLUE:
    offset_y_ind = -1
elif color == ObjectColor.RED:
    offset_y_ind = 0
else:
    offset_y_ind = 1

# Going to place the object
next_place_pose = place_pose.copy_with_offsets(x_offset=offset_x_ind * offset_size,
                                               y_offset=offset_y_ind * offset_size)
robot.pick_place.place_from_pose(next_place_pose)
```

We increment the count_dict dictionary and reset try_without_success

```
count_dict[color] += 1
try_without_success = 0
```

Once the target catch number is achieved, we go to sleep

```
robot.arm.go_to_sleep()
```

Sorting Pick with Conveyor

An interesting way to bring objects to the robot, is the use of a Conveyor Belt. In this examples, we will see how to catch only a certain type of object by stopping the conveyor as soon as the object is detected on the workspace

```
# Initializing variables
offset_size = 0.05
max_catch_count = 4
shape_expected = ObjectShape.CIRCLE
color_expected = ObjectColor.RED

conveyor_id = robot.conveyor.set_conveyor()

catch_count = 0
while catch_count < max_catch_count:
    # Turning conveyor on
    robot.conveyor.run_conveyor(conveyor_id)
    # Moving to observation pose
    robot.arm.move_pose(observation_pose)
    # Check if object is in the workspace
    obj_found, pos_array, shape, color = robot.vision.detect_object(workspace_name,
                                                                    shape=shape_expected,
                                                                    color=color_expected)
    if not obj_found:
        robot.wait(0.5)  # Wait to let the conveyor turn a bit
        continue
    # Stopping conveyor
    robot.conveyor.stop_conveyor(conveyor_id)
    # Making a vision pick
    obj_found, shape, color = robot.visionvision_pick(workspace_name,
                                                      shape=shape_expected,
                                                      color=color_expected)
    if not obj_found:  # If visual pick did not work
        continue

    # Calculate place pose and going to place the object
    next_place_pose = place_pose.copy_with_offsets(x_offset=catch_count * offset_size)
    robot.pick_place.place_from_pose(next_place_pose)

    catch_count += 1

# Stopping & unsetting conveyor
robot.conveyor.stop_conveyor(conveyor_id)
robot.conveyor.unset_conveyor(conveyor_id)

robot.arm.go_to_sleep()
```

Code Details - Sort Picking

Firstly, we initialize your process : we want the robot to catch 4 Red Circles. To do so, we set variables shape_expected and color_expected with **ObjectShape.CIRCLE** and **ObjectColor.RED**

```
offset_size = 0.05
max_catch_count = 4
shape_expected = ObjectShape.CIRCLE
color_expected = ObjectColor.RED
```

We activate the connection with the conveyor and start a loop until the robot has caught max_catch_count objects

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
conveyor_id = robot.set_conveyor()

catch_count = 0
while catch_count < max_catch_count:
```

For each iteration, we firstly run the conveyor belt (if the later is already running, nothing will happen), then go to the observation pose

```
# Turning conveyor on
robot.conveyor.run_conveyor(conveyor_id)
# Moving to observation pose
robot.arm.move_pose(observation_pose)
```

We then check if an object corresponding to our criteria is in the workspace. If not, we wait 0.5 second and then, start a new iteration

```
obj_found, pos_array, shape, color = robot.vision.detect_object(workspace_name,
                                                                shape=shape_expected,
                                                                color=color_expected)
if not obj_found:
    robot.wait(0.5)  # Wait to let the conveyor turn a bit
    continue
```

Else, stop the conveyor and try to make a vision pick

```
# Stopping conveyor
robot.conveyor.stop_conveyor(conveyor_id)
# Making a vision pick
obj_found, shape, color = robot.vision.vision_pick(workspace_name,
                                                   shape=shape_expected,
                                                   color=color_expected)
if not obj_found:  # If visual pick did not work
    continue
```

If Vision Pick succeed, compute new place pose, and place the object

```
# Calculate place pose and going to place the object
next_place_pose = place_pose.copy_with_offsets(x_offset=catch_count * offset_size)
robot.pick_place.place_from_pose(next_place_pose)

catch_count += 1
```

Once the target catch number is achieved, we stop the conveyor and go to sleep

```
# Stopping & unsetting conveyor
robot.conveyor.stop_conveyor(conveyor_id)
robot.conveyor.unset_conveyor(conveyor_id)

robot.arm.go_to_sleep()
```

Examples: Dynamic frames

This document shows how to use dynamic frames.

If you want to see more about dynamic frames functions, you can look at PyNiryo - Frames (index.html#frames)

Danger

If you are using the real robot, make sure the environment around it is clear.

Simple dynamic frame control

This example shows how to create a frame and do a small pick and place in this frame:

```python
from pyniryo2 import *

robot_ip_address = "192.168.1.91"
gripper_speed = 400


if __name__ == '__main__':
    robot = NiryoRobot(robot_ip_address)

    # Create frame
    point_o = [0.15, 0.15, 0]
    point_x = [0.25, 0.2, 0]
    point_y = [0.2, 0.25, 0]

    robot.frames.save_dynamic_frame_from_points("dynamic_frame", "description", point_o, point_x, point_y)

    # Get list of frames
    print(robot.frames.get_saved_dynamic_frame_list())
    # Check creation of the frame
    info = robot.frames.get_saved_dynamic_frame("dynamic_frame")
    print(info)

    # Pick
    robot.tool.update_tool()
    robot.tool.open_gripper(gripper_speed)
    # Move to the frame
    initial_pose = PoseObject(0, 0, 0, 0, 1.57, 0)
    robot.arm.move_pose(initial_pose, "dynamic_frame")
    robot.tool.close_gripper(gripper_speed)

    # Move in frame
    robot.arm.move_linear_relative([0, 0, 0.1, 0, 0, 0], "dynamic_frame")
    robot.arm.move_relative([0.1, 0, 0, 0, 0, 0], "dynamic_frame")
    robot.arm.move_linear_relative([0, 0, -0.1, 0, 0, 0], "dynamic_frame")

    # Place
    robot.tool.open_gripper(gripper_speed)
    robot.arm.move_linear_relative([0, 0, 0.1, 0, 0, 0], "dynamic_frame")

    # Home
    robot.arm.move_joints([0, 0.5, -1.25, 0, 0, 0])

    # Delete frame
    robot.frames.delete_saved_dynamic_frame("dynamic_frame")

    robot.end()
```

Code templates

As code structures are always the same, we wrote down few templates for you to start your code file with a good form

The short template

Very simple, straightforward

```python
from pyniryo2 import *

# Connect to robot & calibrate
robot = NiryoRobot(<robot_ip_address>)
robot.arm.calibrate_auto()

# --- --------- --- #
# --- YOUR CODE --- #
# --- --------- --- #
```

Advanced template

This template let the user defined his own process but it handles connection, calibration, tool equipping, and make the robot go to sleep at the end

```python
from pyniryo2 import *

local_mode = False # Or True
tool_used = ToolID.GRIPPER_1
# Set robot address
robot_ip_address_rpi = "x.x.x.x"
robot_ip_address_local = "127.0.0.1"

robot_ip_address = robot_ip_address_local if local_mode else robot_ip_address_rpi


def process(niryo_edu):
    # --- --------- --- #
    # --- YOUR CODE --- #
    # --- --------- --- #

if __name__ == '__main__':
    # Connect to robot
    robot = NiryoRobot(robot_ip_address)
    # Calibrate robot if robot needs calibration
    robot.arm.calibrate_auto()
    # Equip tool
    robot.tool.update_tool()
    # Launching main process
    process(client)
    # Ending
    robot.arm.go_to_sleep()
```

Advanced template for conveyor

Same as Advanced template but with a conveyor

```python
from pyniryo2 import *

# Set robot address
robot_ip_address = "x.x.x.x"


def process(robot, conveyor_id):
    robot.conveyor.run_conveyor(conveyor_id)

    # --- --------- --- #
    # --- YOUR CODE --- #
    # --- --------- --- #

    robot.conveyor.stop_conveyor()


if __name__ == '__main__':
    # Connect to robot
    robot = NiryoRobot(robot_ip_address)
    # Calibrate robot if robot needs calibration
    robot.arm.calibrate_auto()
    # Equip tool
    robot.tool.update_tool()
    # Activating connexion with conveyor
    conveyor_id = robot.conveyor.set_conveyor()
    # Launching main process
    process(robot, conveyor_id)
    # Ending
    robot.arm.go_to_sleep()
    # Deactivating connexion with conveyor
    robot.conveyor.unset_conveyor(conveyor_id)
```

Advanced template for vision

Huge template for vision users !

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```python
from pyniryo2 import *

local_mode = False # Or True
workspace_name = "workspace_1"  # Robot's Workspace Name
# Set robot address
robot_ip_address_rpi = "x.x.x.x"
robot_ip_address_local = "127.0.0.1"

robot_ip_address = robot_ip_address_local if local_mode else robot_ip_address_rpi

# The pose from where the image processing happens
observation_pose = PoseObject(
    x=0.18, y=0.0, z=0.35,
    roll=0.0, pitch=1.57, yaw=-0.2,
)

# Center of the conditioning area
place_pose = PoseObject(
    x=0.0, y=-0.23, z=0.12,
    roll=0.0, pitch=1.57, yaw=-1.57
)

def process(robot):
    robot.arm.move_pose(observation_pose)
    catch_count = 0
    while catch_count < 3:
        ret = robot.vision.get_target_pose_from_cam(workspace_name,
                                                    height_offset=0.0,
                                                    shape=ObjectShape.ANY,
                                                    color=ObjectColor.ANY)
        obj_found, obj_pose, shape, color = ret
        if not obj_found:
            continue
        catch_count += 1
        # --- --------- --- #
        # --- YOUR CODE --- #
        # --- --------- --- #
        robot.pick_place.place_from_pose(place_pose)

if __name__ == '__main__':
    # Connect to robot
    robot = NiryoRobot(robot_ip_address)
    # Calibrate robot if robot needs calibration
    robot.arm.calibrate_auto()
    # Equip tool
    robot.tool.update_tool()
    # Launching main process
    process(client)
    # Ending
    robot.arm.go_to_sleep()
```

Callbacks Templates

Template for event integration !

```
# Imports
from pyniryo2 import *
from threading import Event

robot_ip = "xxx.xxx.xxx.xxx"
robot_ip_address_local = "127.0.0.1"

# Events
update_tool_event = Event()
update_tool_event.clear()

calibrated_event = Event()
calibrated_event.clear()

# Poses
pose_1 = PoseObject()

pose_2 = PoseObject()

# Callbacks
def update_tool_success_callback(result):
    update_tool_event.set()
    print 'Update Tool: ', result['message']

def update_tool_error_callback(result):
    print 'Update Tool: ', result['message']

def calibrate_success_callback(result):
    calibrated_event.set()
    print 'Calibrate Callback: ', result["message"]

def calibrate_error_callback(result):
    print 'Calibrate Callback: ', result["message"]

    """
    Add Callbacks Here
    """

def action_function(robot):

    """
    Don't put niryo_robot in parameter, it will take the pyniryo2 package
    add your function here
    """


if __name__ == "__main__":

    # Connect to robot
    robot = NiryoRobot(robot_ip)

    # Calibrate robot if robot needs calibration
    robot.arm.calibrate_auto(callback=calibrate_success_callback, errback=calibrate_error_callback)
    calibrated_event.wait(20)
    if not calibrated_event.is_set():
        quit

    robot.tool.update_tool(callback=update_tool_success_callback, errback=update_tool_error_callback)
    update_tool_event.wait()

    action_function(robot)

    robot.arm.go_to_sleep()
```

API Documentation

Master controls with PyNiryo2 with full detailed functions here (index.html#document-source/api_doc/niryo_robot)

NiryoRobot

The NiryoRobot class includes the different APIs of the PyNiryo2 library. It allows the connection of the program to the robot via roslibpy. This interface facilitates and centralizes all the control functions of the Niryo environment and products.

NiryoRobot - Command functions

This section reference all existing functions of the NiryoRobot client, which include

- Connecting to your Ned
- Disconnecting from your Ned
- Waiting
- Access to the entire PyNiryo2 API

- **NiryoRobot**

All functions to control the robot are accessible via an instance of the classNiryoRobot

```
robot = NiryoRobot(<robot_ip_address>)

robot.run("10.10.10.10")
robot.wait(2) # wait 2 seconds
robot.end()
```

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

See examples on Examples Section (index.html#examples-basics)

List of functions subsections:

- NiryoRobot functions
- NiryoRobot properties

NiryoRobot functions

class **NiryoRobot**(*ip_address='127.0.0.1'*, *port=9090*)

Connect your robot to your computer:

```
robot_simulation = NiryoRobot("127.0.0.1") # Simulation

robot_hotpot = NiryoRobot("10.10.10.10") # Hotspot

robot_ethernet = NiryoRobot("169.254.200.201") # Ethernet
```

Parameters:

- **ip_address** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – robot ROS ip
- **port** (*int* (https://docs.python.org/3/library/functions.html#int)) – robot ROS port

property **client**

Get the Niryo Ros client

Returns:
NiryoRos client

Return type:
NiryoRos (index.html#pyniryo2.niryo_ros.NiryoRos)

end()

Disconnect from your robot and ROS:

```
# Start
robot = NiryoRobot("10.10.10.10")

# End
robot.end()
```

Return type:
None (https://docs.python.org/3/library/constants.html#None)

static **wait**(*duration*)

Wait for a certain time

Parameters:
duration (*float* (https://docs.python.org/3/library/functions.html#float)) – duration in seconds

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot properties

class **NiryoRobot**(*ip_address='127.0.0.1'*, *port=9090*)

Connect your robot to your computer:

```
robot_simulation = NiryoRobot("127.0.0.1") # Simulation

robot_hotpot = NiryoRobot("10.10.10.10") # Hotspot

robot_ethernet = NiryoRobot("169.254.200.201") # Ethernet
```

Parameters:

- **ip_address** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – robot ROS ip
- **port** (*int* (https://docs.python.org/3/library/functions.html#int)) – robot ROS port

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

property **arm**

Access to the Arm API

Example:

```
robot = NiryoRobot(<robot_ip_address>)
robot.arm.calibrate_auto()
robot.arm.move_joints([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
```

Return type:
Arm (index.html#pyniryo2.arm.arm.Arm)

property **conveyor**

Access to the Conveyor API

Example:

```
robot = NiryoRobot(<robot_ip_address>)
conveyor_id = robot.conveyor.set_conveyor()
robot.conveyor.run_conveyor(conveyor_id)
```

Return type:
Conveyor (index.html#pyniryo2.conveyor.conveyor.Conveyor)

property **io**

Access to the I/Os API

Example:

```
robot = NiryoRobot(<robot_ip_address>)
robot.io.set_pin_mode(PinID.GPIO_1A, PinMode.INPUT)
robot.io.digital_write(PinID.GPIO_1A, PinState.HIGH)
```

Return type:
IO (index.html#pyniryo2.io.io.IO)

property **pick_place**

Access to the PickPlace API

Example:

```
robot = NiryoRobot(<robot_ip_address>)
robot.pick_place.pick_from_pose([0.2, 0.0, 0.1, 0.0, 1.57, 0.0])
robot.pick_place.place_from_pose([0.0, 0.2, 0.1, 0.0, 1.57, 0.0])
```

Return type:
PickPlace (index.html#pyniryo2.pick_place.pick_place.PickPlace)

property **saved_poses**

Access to the SavedPoses API

Example:

```
robot = NiryoRobot(<robot_ip_address>)
pose_name_list = robot.saved_poses.get_saved_pose_list()
robot.saved_poses.get_pose_saved(pose_name_list[0])
```

Return type:
SavedPoses (index.html#pyniryo2.saved_poses.saved_poses.SavedPoses)

property **sound**

Access to the Sound API

Example:

```
robot = NiryoRobot(<robot_ip_address>)
sound.play_sound_user("test_sound.wav")
sound_name = sound.get_sounds()[0]
sound_duration = sound.play(sound_name)
```

Return type:
Sound (index.html#pyniryo2.sound.Sound)

property **tool**

Access to the Tool API

Example:

```
robot = NiryoRobot(<robot_ip_address>)
robot.tool.update_tool()
robot.tool.grasp_with_tool()
robot.tool.release_with_tool()
```

Return type:
Tool (index.html#pyniryo2.tool.tool.Tool)

property **trajectories**

Access to the Trajectories API

Example:

```
robot = NiryoRobot(<robot_ip_address>)
trajectories = robot.trajectories.get_saved_trajectory_list()
if len(trajectories) > 0:
    robot.trajectories.execute_trajectory_saved(trajectories[0])
```

Return type:
Trajectories (index.html#pyniryo2.trajectories.trajectories.Trajectories)

property **vision**

Access to the Vision API

Example:

```
robot = NiryoRobot(<robot_ip_address>)
robot.vision.vision_pick("workspace_1", 0.0, ObjectShape.ANY, ObjectColor.ANY)
robot.vision.detect_object("workspace_1", ObjectShape.ANY, ObjectColor.ANY)
```

Return type:
Vision (index.html#pyniryo2.vision.vision.Vision)

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

property **led_ring**

Access to the Led Ring API

Example:

```
robot = NiryoRobot(<robot_ip_address>)
niryo_robot.led_ring.led_ring_flash([20,255,78], iterations = 10, wait = True, frequency = 8)
niryo_robot.led_ring.led_ring_turn_off()
```

Return type:
LedRing (index.html#pyniryo2.led_ring.LedRing)

property **frames**

Access to the frame API

Example:

```
robot = NiryoRobot(<robot_ip_address>)
```

Return type:
Frames (index.html#pyniryo2.frames.frames.Frames)

Globals Enums

List of enums:

- **RobotErrors**
- **ArmMoveCommandType**

class **RobotErrors**(*value*)

An enumeration.

SUCCESS= *1*

CANCELLED= *2*

PREEMPTED= *3*

FAILURE= *-1*

ABORTED= *-3*

STOPPED= *-4*

ROS_ERROR= *-20*

FILE_ALREADY_EXISTS= *-30*

UNKNOWN_COMMAND= *-50*

NOT_IMPLEMENTED_COMMAND= *-51*

INVALID_PARAMETERS= *-52*

HARDWARE_FAILURE= *-110*

HARDWARE_NOT_OK= *-111*

LEARNING_MODE_ON= *-112*

CALIBRATION_NOT_DONE= *-113*

DIGITAL_IO_PANEL_ERROR= *-114*

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

LED_MANAGER_ERROR= *-115*

BUTTON_ERROR= *-116*

WRONG_MOTOR_TYPE= *-117*

DXL_WRITE_ERROR= *-118*

DXL_READ_ERROR= *-119*

CAN_WRITE_ERROR= *-120*

CAN_READ_ERROR= *-121*

NO_CONVEYOR_LEFT= *-122*

NO_CONVEYOR_FOUND= *-123*

CONVEYOR_ID_INVALID= *-124*

CALIBRATION_IN_PROGRESS= *-125*

VIDEO_STREAM_ON_OFF_FAILURE= *-170*

VIDEO_STREAM_NOT_RUNNING= *-171*

OBJECT_NOT_FOUND= *-172*

MARKERS_NOT_FOUND= *-173*

ARM_COMMANDER_FAILURE= *-220*

GOAL_STILL_ACTIVE= *-221*

JOG_CONTROLLER_ENABLED= *-222*

CONTROLLER_PROBLEMS= *-223*

SHOULD_RESTART= *-224*

JOG_CONTROLLER_FAILURE= *-225*

PLAN_FAILED= *-230*

NO_PLAN_AVAILABLE= *-231*

INVERT_KINEMATICS_FAILURE= *-232*

TOOL_FAILURE= *-251*

TOOL_ID_INVALID= *-252*

TOOL_NOT_CONNECTED= *-253*

TOOL_ROS_INTERFACE_ERROR= *-254*

POSES_HANDLER_CREATION_FAILED= *-300*

POSES_HANDLER_REMOVAL_FAILED= *-301*

POSES_HANDLER_READ_FAILURE= *-302*

POSES_HANDLER_COMPUTE_FAILURE= *-303*

WORKSPACE_CREATION_FAILED= *-308*

PROGRAMS_MANAGER_FAILURE= *-320*

PROGRAMS_MANAGER_READ_FAILURE= *-321*

PROGRAMS_MANAGER_UNKNOWN_LANGUAGE= *-325*

PROGRAMS_MANAGER_NOT_RUNNABLE_LANGUAGE= *-326*

PROGRAMS_MANAGER_EXECUTION_FAILED= *-327*

PROGRAMS_MANAGER_STOPPING_FAILED= *-328*

PROGRAMS_MANAGER_AUTORUN_FAILURE= *-329*

PROGRAMS_MANAGER_WRITING_FAILURE= *-330*

PROGRAMS_MANAGER_FILE_ALREADY_EXISTS= *-331*

PROGRAMS_MANAGER_FILE_DOES_NOT_EXIST= *-332*

SERIAL_FILE_ERROR= *-400*

SERIAL_UNKNOWN_ERROR= *-401*

MQTT_PUBLISH_FUNCTION_DOESNT_EXIST= *-420*

MQTT_PUBLISH_FUNCTION_INVALID_ARGUMENTS= *-421*

SYSTEM_API_CLIENT_UNKNOWN_ERROR= *-440*

SYSTEM_API_CLIENT_INVALID_ROBOT_NAME= *-441*

SYSTEM_API_CLIENT_REQUEST_FAILED= *-442*

class **ArmMoveCommandType**(*value*)

Enumeration of Arm Move Command : it used for move commands

JOINTS= *0*

POSE= *1*

POSITION= *2*

RPY= *3*

POSE_QUAT= *4*

LINEAR_POSE= *5*

SHIFT_POSE= *6*

SHIFT_LINEAR_POSE= *7*

EXECUTE_TRAJ= *8*

DRAW_SPIRAL= *9*

DRAW_CIRCLE= *10*

EXECUTE_FULL_TRAJ= *11*

EXECUTE_RAW_TRAJ= *12*

Globals Objects

- **PoseObject** (index.html#pyniryo2.objects.PoseObject)

class **PoseObject**(*x*, *y*, *z*, *roll*, *pitch*, *yaw*)

Pose object which stores x, y, z, roll, pitch & yaw parameters

Variables:
- **x** (*float* (https://docs.python.org/3/library/functions.html#float)) – X (meter)
- **y** (*float* (https://docs.python.org/3/library/functions.html#float)) – Y (meter)
- **z** (*float* (https://docs.python.org/3/library/functions.html#float)) – Z (meter)
- **roll** (*float* (https://docs.python.org/3/library/functions.html#float)) – Roll (radian)
- **pitch** (*float* (https://docs.python.org/3/library/functions.html#float)) – Pitch (radian)
- **yaw** (*float* (https://docs.python.org/3/library/functions.html#float)) – Yaw (radian)

copy_with_offsets(*x_offset=0.0*, *y_offset=0.0*, *z_offset=0.0*, *roll_offset=0.0*, *pitch_offset=0.0*, *yaw_offset=0.0*)

Create a new pose from copying from copying actual pose with offsets

Return type:
PoseObject (index.html#pyniryo2.objects.PoseObject)

to_list()

Return a list [x, y, z, roll, pitch, yaw] corresponding to the pose's parameters

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

property **quaternion**

Return the quaternion in a list [qx, qy, qz, qw]

Returns:
quaternion [qx, qy, qz, qw]

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float)]

property **quaternion_pose**

Return the position and the quaternion in a list [x, y, z, qx, qy, qz, qw]

Returns:
position [x, y, z] + quaternion [qx, qy, qz, qw]

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float)]

static **euler_to_quaternion**(*roll*, *pitch*, *yaw*)

Convert euler angles to quaternion

Parameters:
- **roll** (*float* (https://docs.python.org/3/library/functions.html#float)) – roll in radians
- **pitch** (*float* (https://docs.python.org/3/library/functions.html#float)) – pitch in radians
- **yaw** (*float* (https://docs.python.org/3/library/functions.html#float)) – yaw in radians

Returns:
quaternion in a list [qx, qy, qz, qw]

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float)]

static **quaternion_to_euler_angle**(*qx, qy, qz, qw*)

Convert euler angles to quaternion

Parameters:
- **qx** (*float* (https://docs.python.org/3/library/functions.html#float)) –
- **qy** (*float* (https://docs.python.org/3/library/functions.html#float)) –
- **qz** (*float* (https://docs.python.org/3/library/functions.html#float)) –
- **qw** (*float* (https://docs.python.org/3/library/functions.html#float)) –

Returns:
euler angles in a list [roll, pitch, yaw]

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float)]

NiryoTopic

Pyniryo2 is based on the python library roslibpy to collect information from the robot. This information is sent by ROS via topics. This class is an overlay of the API roslibpy Topic (https://roslibpy.readthedocs.io/en/latest/reference/index.html#topics). It allows you to subscribe to a topic to collect the information from the topic as soon as it is published, or ask for only one value. Please refer to the Niryo robot ROS doc to see the compatible topics.

NiryoTopic - Usage

Here is a simple example of using the class without conversion:

```
>> robot = NiryoRobot(<robot_ip_address>)
>> client = robot.client
>> joint_states_topic = NiryoTopic(client, '/joint_states', 'sensor_msgs/JointState')
>> joint_states_topic()
{u'header': {u'stamp': {u'secs': 1626092430, u'nsecs': 945618510}, u'frame_id': u'', u'seq': 13699},
u'position': [0.0, 0.6, -1.3, 0.0, 0.0, 0.0], u'effort': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
u'name': [u'joint_1', u'joint_2', u'joint_3', u'joint_4', u'joint_5', u'joint_6'],
u'velocity': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]}
```

Here is a simple example of using the class with conversion:

```
>> def joint_states_topic_conversion(msg):
       return msg["position"]

>> robot = NiryoRobot(<robot_ip_address>)
>> client = robot.client
>> joint_states_topic = NiryoTopic(client, '/joint_states', 'sensor_msgs/JointState', joint_states_topic_conversion)
>> joint_states_topic()
[0.0, 0.6, -1.3, 0.0, 0.0, 0.0]
>> joint_states_topic.value
[0.0, 0.6, -1.3, 0.0, 0.0, 0.0]
```

Here is a simple example of using the class with a callback:

```
def joint_states_topic_conversion(msg):
    return msg["position"]

def joint_states_callback(msg):
    print(msg)  # print the list of joints position

robot = NiryoRobot("127.0.0.1")
client = robot.client
joint_states_topic = NiryoTopic(client, '/joint_states', 'sensor_msgs/JointState', joint_states_topic_conversion)
joint_states_topic.subscribe(joint_states_callback)

...

joint_states_topic.unsubscribe()
```

NiryoTopic - Class

- **NiryoTopic**

class **NiryoTopic**(*client, topic_name, topic_type, conversion_function=None, timeout=3*)

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Represent a Ros Topic instance. It supports both the request of a single value and/or callbacks. This class is a wrapper of roslibpy Topic instance (https://roslibpy.readthedocs.io/en/latest/reference/index.html#topics (https://roslibpy.readthedocs.io/en/latest/reference/index.html#topics))

Parameters:

- **client** (*roslibpy.Ros*) – Instance of the ROS connection.
- **topic_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – Topic name.
- **topic_type** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – Topic type.
- **conversion_function** (*function*) – convert the response of the topic in a specific type.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout while waiting a message.

property **is_subscribed**

Return the topic connection status.

Returns:
True if already subscribed, False otherwise.

Return type:
Bool

property **value**

Return the last value of the topic.

Returns:
The last value of the topic. The value depends on the conversion function of the topic. By default, it will be a dict.

Return type:

subscribe(*callback*)

Subscribe a callback to the topic. A TopicException will be thrown if the topic is already subscribed.

Parameters:
callback (*function(dict* (https://docs.python.org/3/library/stdtypes.html#dict)*,)*) – The callback function which is called at each incoming topic message.

Returns:
None

Return type:
None (https://docs.python.org/3/library/constants.html#None)

unsubscribe()

Unsubscribe to the topic.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

publish(*msg*)

Publish a message on the topic

Parameters:
msg (*dict of roslibpy.Message*) – jsonified topic message content

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRos

The NiryoRos class inherits the attributes and functions of roslibpy.Ros. It allows the connection of the program to the ROS Master of the robot via roslibpy. It adds an overlay to manage the versions of the robot and a recognition of the API by the robot which is necessary for the use of certain functions.

class **NiryoRos**(*ip_address='127.0.0.1', port=9090*)

Connect to your computer to ros:

```
ros_instance = NiryoRos("127.0.0.1") # Simulation

ros_instance = NiryoRos("10.10.10.10") # Hotspot

ros_instance = NiryoRos("169.254.200.201") # Ethernet
```

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Based on the roslibpy ROS client: https://roslibpy.readthedocs.io/en/latest/reference/index.html#roslibpy.Ros (https://roslibpy.readthedocs.io/en/latest/reference/index.html#roslibpy.Ros)

Parameters:

- **ip_address** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – ip of the ros master
- **port** (*int* (https://docs.python.org/3/library/functions.html#int)) – usually 9090

close()

Disconnect from ROS.

terminate()

Signals the termination of the main event loop.

property **hardware_version**

Get the hardware version of the robot (one, ned, ned2)

Returns:
The hardware version of the robot (one, ned, ned2)

Return type:
str (https://docs.python.org/3/library/stdtypes.html#str)

exception **NiryoRosTimeoutException**

Arm

This file presents the different Arm - Command functions, Arm - Enums, Arm - Niryo Topics & Arm - Objects available with the Arm API

Arm - Command functions

This section reference all existing functions to control your robot arm, which include

- Getting the robot state
- Moving the arm
- Getting inverse and forward kinematics
- Calibrating the robot

All functions to control the robot are accessible via an instance of the classNiryoRobot (index.html#niryorobot)

```
robot = NiryoRobot(<robot_ip_address>)

robot.arm.calibrate_auto()
robot.arm.move_joints([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
...
```

See examples on Examples Section (index.html#examples-basics)

List of functions subsections:

- Calibration functions
- Robot move functions
- Robot status functions
- Learning mode functions
- Kinematics functions

class **Arm**(*client*)

Arm robot functions

Example:

```
ros_instance = NiryoRos("10.10.10.10") # Hotspot
arm_interface = Arm(ros_instance)
```

Parameters:
client (*NiryoRos* (index.html#pyniryo2.niryo_ros.NiryoRos)) – Niryo ROS client

Calibration functions

Arm.calibrate(*calibrate_mode*, *callback=None*, *errback=None*, *timeout=None*)

Calibrates (manually or automatically) motors. Automatic calibration will do nothing if motors are already calibrated

Examples:

```
# Synchronous use
arm.calibrate(CalibrateMode.MANUAL)
arm.calibrate(CalibrateMode.AUTO)

# Asynchronous use
def calibration_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Calibration failed")
    else:
        print("Calibration completed with success")

arm.calibrate(CalibrateMode.AUTO, calibration_callback)
```

Parameters:

- **calibrate_mode** (*CalibrateMode*) – Auto or Manual
- **callback** (*function*) – Callback invoked on successful execution.
- **errback** (*function*) – Callback invoked on error.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:

None (https://docs.python.org/3/library/constants.html#None)

Arm.calibrate_auto(*callback=None, errback=None, timeout=None*)

Starts a automatic motors calibration if motors are not calibrated yet.

Examples:

```
# Synchronous use
arm.calibrate_auto()

# Asynchronous use
def calibration_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Calibration failed")
    else:
        print("Calibration completed with success")

arm.calibrate_auto(calibration_callback)
```

Parameters:

- **callback** (*function*) – Callback invoked on successful execution.
- **errback** (*function*) – Callback invoked on error.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:

None (https://docs.python.org/3/library/constants.html#None)

Arm.request_new_calibration(*callback=None, errback=None, timeout=None*)

Starts a automatic motors calibration even if motors are calibrated yet.

Examples:

```
# Synchronous use
arm.request_new_calibration()

# Asynchronous use
def calibration_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Calibration failed")
    else:
        print("Calibration completed with success")

arm.request_new_calibration(calibration_callback)
```

Parameters:

- **callback** (*function*) – Callback invoked on successful execution.
- **errback** (*function*) – Callback invoked on error.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:

None (https://docs.python.org/3/library/constants.html#None)

Arm.reset_calibration(*timeout=2*)

Resets current calibration status. A new calibration is then necessary.

Parameters:

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

timeout (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Arm.need_calibration()

Returns a bool indicating whereas the robot motors need to be calibrate.

Returns:
True if calibration is needed, False otherwise.

Return type:
bool (https://docs.python.org/3/library/functions.html#bool)

Robot move functions

Arm.set_arm_max_velocity(*percentage_speed*)

Limit arm max velocity to a percentage of its maximum velocity

Parameters:
percentage_speed (*int* (https://docs.python.org/3/library/functions.html#int)) – Should be between 1 & 100

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Arm.go_to_sleep()

Go to home pose and activate learning mode

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Arm.stop_move(*callback=None*, *errback=None*, *timeout=None*)

Stop a current execution of move_pose, move_joint or move_linear_pose. The robot will stop at its current position . If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

Examples:

```
# Synchronous use
arm.stop_move()

# Asynchronous use
def stop_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Succeeded")
    else:
        print("Failed")

arm.stop_move(stop_callback)
```

Parameters:
- **callback** (*function*) – Callback invoked on successful execution.
- **errback** (*function*) – Callback invoked on error.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Arm.move_to_home_pose(*callback=None*)

Move to a position where the forearm lays on shoulder If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

Parameters:
callback (*function*) – Callback invoked on successful execution.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Arm.move_joints(*joints*, *callback=None*)

Move robot joints. Joints are expressed in radians. If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

All lines of the next example realize the same operation:

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
arm.joints = [0.2, 0.1, 0.3, 0.0, 0.5, 0.0]
arm.move_joints([0.2, 0.1, 0.3, 0.0, 0.5, 0.0])

def move_callback(_):
    print("Move completed")

arm.move_joints([0chronous use
arm.calibrate(CalibrateMode.MANUAL)
arm.calibrate(CalibrateMode.AUTO)

# Asynchronous use
def calibration_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Calibration failed")
    else:
        print("Calibration completed with success")

arm.calibrate(CalibrateMode.AUTO, calibration_callback)
```

Parameters:

- **callback** (*function*) – Callback invoked on successful execution.
- **joints** (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – a list of 6 joints

Return type:

None (https://docs.python.org/3/library/constants.html#None)

Arm.move_pose(*pose, frame='', callback=None*)

Move robot end effector pose to a (x, y, z, roll, pitch, yaw) pose in the frame (frame_name) if defined. x, y & z are expressed in meters / roll, pitch & yaw are expressed in radians If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

All lines of the next example realize the same operation:

```
arm.pose = [0.2, 0.1, 0.3, 0.0, 0.5, 0.0]
arm.move_pose([0.2, 0.1, 0.3, 0.0, 0.5, 0.0])
arm.move_pose(PoseObject(0.2, 0.1, 0.3, 0.0, 0.5, 0.0))
arm.move_pose([0.2, 0.1, 0.3, 0.0, 0.5, 0.0], "default_frame")
arm.move_pose(PoseObject(0.2, 0.1, 0.3, 0.0, 0.5, 0.0), "default_frame")

def move_callback(_):
    print("Move completed")

arm.move_joints([0.2, 0.1, 0.3, 0.0, 0.5, 0.0], callback=move_callback)
```

Parameters:

- **callback** (*function*) – Callback invoked on successful execution.
- **pose** (*Union[tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#pyniryo2.objects.PoseObject)*]*) – either a list of 6 coordinates or a PoseObject
- **frame** – name of the frame

Return type:

None (https://docs.python.org/3/library/constants.html#None)

Arm.move_linear_pose(*pose, frame='', callback=None*)

Move robot end effector pose to a (x, y, z, roll, pitch, yaw) pose in a linear way, in the frame (frame_name) if defined. If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

All lines of the next example realize the same operation:

```
arm.move_linear_pose([0.2, 0.1, 0.3, 0.0, 0.5, 0.0])
arm.move_linear_pose(PoseObject(0.2, 0.1, 0.3, 0.0, 0.5, 0.0))
arm.move_linear_pose([0.2, 0.1, 0.3, 0.0, 0.5, 0.0], "default_frame")
arm.move_linear_pose(PoseObject(0.2, 0.1, 0.3, 0.0, 0.5, 0.0), "default_frame")

def move_callback(_):
    print("Move completed")

arm.move_linear_pose([0.2, 0.1, 0.3, 0.0, 0.5, 0.0], callback=move_callback)
```

Parameters:

- **callback** (*function*) – Callback invoked on successful execution.
- **pose** (*Union[tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#pyniryo2.objects.PoseObject)*]*) – either or a list of 6 coordinates or a PoseObject
- **frame** – name of the frame

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Arm.shift_pose(*axis, shift_value, callback=None*)

Shift robot end effector pose along one axis If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

Examples:

```
self.arm.shift_pose(RobotAxis.X, 0.05)
self.arm.shift_pose(RobotAxis.Y, -0.05)
self.arm.shift_pose(RobotAxis.Z, 0.1)
self.arm.shift_pose(RobotAxis.ROLL, 1.57)
self.arm.shift_pose(RobotAxis.PITCH, -1.57)
self.arm.shift_pose(RobotAxis.YAW, 0.78)

def move_callback(_):
    print("Move completed")

self.arm.shift_pose(RobotAxis.X, 0.1, move_callback)
```

Parameters:
- **axis** (*RobotAxis*) – Axis along which the robot is shifted
- **shift_value** (*float* (https://docs.python.org/3/library/functions.html#float)) – In meter for X/Y/Z and radians for roll/pitch/yaw
- **callback** (*function*) – Callback invoked on successful execution.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Arm.move_relative(*offset, frame='world'*)

Move robot end of a offset in a frame

Example:

```
arm.move_relative([0.05, 0.05, 0.05, 0.3, 0, 0], "default_frame")
```

Parameters:
- **offset** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – list which contains offset of x, y, z, roll, pitch, yaw
- **frame** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of local frame

Returns:
status, message

Return type:
(int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

Arm.move_linear_relative(*offset, frame='world'*)

Move robot end of a offset by a linear movement in a frame

Example:

```
arm.move_linear_relative([0.05, 0.05, 0.05, 0.3, 0, 0], "default_frame")
```

Parameters:
- **offset** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – list which contains offset of x, y, z, roll, pitch, yaw
- **frame** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of local frame

Returns:
status, message

Return type:
(int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

Arm.set_jog_control(*enabled*)

Set jog control mode if param is True, else turn it off

Parameters:
enabled (*Bool*) – True or False

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Arm.jog_joints(*joints_offset, callback=None, errback=None, timeout=None*)

Jog robot joints'. Jog corresponds to a shift without motion planning. Values are expressed in radians. If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

Examples:

```
arm.jog_joints([0.1, 0.0, 0.5, 0.0, 0.0, -1.57])

def jog_callback(_):
   print("Jog completed")
   arm.set_jog_control(False) # Disable Jog interface

arm.jog_joints([0.1, 0.0, 0.5, 0.0, 0.0, -1.57], jog_callback)
```

Parameters:

- **joints_offset** (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – a list of 6 joints offset
- **callback** (*function*) – Callback invoked on successful execution.
- **errback** (*function*) – Callback invoked on error.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:

None (https://docs.python.org/3/library/constants.html#None)

Arm.jog_pose(*pose_offset, callback=None, errback=None, timeout=None*)

Jog robot end effector pose Jog corresponds to a shift without motion planning Arguments are [dx, dy, dz, d_roll, d_pitch, d_yaw] dx, dy & dz are expressed in meters / d_roll, d_pitch & d_yaw are expressed in radians If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

Examples:

```
arm.jog_pose([0.01, 0.0, 0.05, 0.0, 0.0, -1.57])

def jog_callback(_):
   print("Jog completed")
   arm.set_jog_control(False) # Disable Jog interface

arm.jog_pose([0.1, 0.0, 0.5, 0.0, 0.0, -1.57], jog_callback)
```

Parameters:

- **pose_offset** (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – a list of 6 offset
- **callback** (*function*) – Callback invoked on successful execution.
- **errback** (*function*) – Callback invoked on error.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:

None (https://docs.python.org/3/library/constants.html#None)

Robot status functions

property **Arm.hardware_status**

Returns the hardware state client which can be used synchronously or asynchronously to obtain the hardware state.

Examples:

```
# Get last value
arm.hardware_status()
arm.hardware_status.value

# Subscribe a callback
def hs_callback(msg):
   print msg.voltage

arm.hardware_status.subscribe(hs_callback)
arm.hardware_status.unsubscribe()
```

Returns:

hardware state topic instance

Return type:

NiryoTopic (index.html#pyniryo2.niryo_topic.NiryoTopic)

property **Arm.joints_state**

Get the joints state topic which can be used synchronously or asynchronously to obtain the joints state. The joints state topic returns a JointStateObject.

It can be used as follows::

```
# Get last joint state
joint_state = arm.joints_state()
joint_state = arm.joints_state.value

joint_names = arm.joints_state().name
joint_positions = arm.joints_state().position
joint_velocities = arm.joints_state.value.velocity

# Raise a callback at each new value
from __future__ import print_function

arm.joints_state.subscribe(lambda message: print(message.position))
arm.joints_state.unsubscribe()
```

Returns:
Joint states topic.

Return type:
NiryoTopic (index.html#pyniryo2.niryo_topic.NiryoTopic)

Arm.get_joints()

Get joints value in radians You can also use a getter

```
joints = arm.get_joints()
joints = arm.joints
```

Returns:
List of joints value

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

property **Arm.joints**

Get joints value in radians

Returns:
List of joints value

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

property **Arm.pose**

Get end effector link pose as [x, y, z, roll, pitch, yaw]. x, y & z are expressed in meters / roll, pitch & yaw are expressed in radians

You can also use a getter

```
pose = arm.pose
pose_list = arm.pose.to_list()
x, y, z, roll, pitch, yaw = arm.pose.to_list()
```

Returns:
end effector link pose

Return type:
PoseObject (index.html#pyniryo2.objects.PoseObject)

property **Arm.get_pose**

Get the end effector link pose topic which can be used synchronously or asynchronously to obtain the end effector link pose. The joints state topic returns a PoseObject. x, y & z are expressed in meters / roll, pitch & yaw are expressed in radians

See below some usage

```
pose = arm.get_pose()
pose = arm.get_pose.value
pose_list = arm.get_pose().to_list()
x, y, z, roll, pitch, yaw = arm.get_pose().to_list()

arm.get_pose.subscribe(callback)
arm.get_pose.unsubscribe()
```

Returns:
end effector link pose topic

Return type:

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

NiryoTopic (index.html#pyniryo2.niryo_topic.NiryoTopic)

Arm.get_pose_quat()

Get end effector link pose in Quaternion coordinates

Returns:
Position and quaternion coordinates concatenated in a list : [x, y, z, qx, qy, qz, qw]

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

Learning mode functions

property **Arm.learning_mode**

Returns the learning mode client which can be used synchronously or asynchronously to obtain the learning mode state. The learning mode client returns a boolean value.

Examples:

```
# Get last value
arm.learning_mode()
if arm.learning_mode.value:
    print("Learning mode enabled"))

# Subscribe a callback
def lm_callback(is_learning_mode_enabled):
    print is_learning_mode_enabled

arm.learning_mode.subscribe(lm_callback)
arm.learning_mode.unsubscribe()
```

Returns:
learning mode state topic instance

Return type:
NiryoTopic (index.html#pyniryo2.niryo_topic.NiryoTopic)

Arm.get_learning_mode()

Get learning mode state.

Returns:
True if learning mode is on

Return type:
bool (https://docs.python.org/3/library/functions.html#bool)

Arm.set_learning_mode(*enabled*)

Set learning mode if param is True , else turn it off

Parameters:
enabled (*bool* (https://docs.python.org/3/library/functions.html#bool)) – True or False

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Kinematics functions

Arm.forward_kinematics(**args*)

Compute forward kinematics of a given joints configuration and give the associated spatial pose

Examples:

```
pose_obj = arm.forward_kinematics(1.57, 0.0, 0.0, 0.78, 0.0, -1.57)
pose_obj = arm.forward_kinematics([1.57, 0.0, 0.0, 0.78, 0.0, -1.57])
```

Parameters:
args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – either 6 args (1 for each joints) or a list of 6 joints

Return type:
PoseObject (index.html#pyniryo2.objects.PoseObject)

Arm.inverse_kinematics(**args*)

Compute inverse kinematics

Examples:

```
joint_list = arm.inverse_kinematics(0.2, 0.0, 0.3, 0.0, 1.57, 0.0)
joint_list = arm.inverse_kinematics([0.2, 0.0, 0.3, 0.0, 1.57, 0.0])
joint_list = arm.inverse_kinematics(PoseObject(0.2, 0.0, 0.3, 0.0, 1.57, 0.0))
```

Parameters:
args (*Union[tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#pyniryo2.objects.PoseObject)*]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a PoseObject

Returns:
List of joints value

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

Arm - Niryo Topics

The use of these functions is explained in the NiryoTopic (index.html#niryotopic) section. They allow the acquisition of data in real time by callbacks or by direct call.

Arm's Niryo Topics

Name	Function	Return type
/joint_states	**joints_state**	**JointStateObject**
/niryo_robot/robot_state	**get_pose**	**PoseObject**
/niryo_robot_hardware_interface/hardware_status	**hardware_status**	**HardwareStatusObject**
/niryo_robot/learning_mode/state	**learning_mode**	**bool** (https://docs.python.org/3/library/functions.html#bool)
/niryo_robot/max_velocity_scaling_factor	**get_arm_max_velocity**	**float** (https://docs.python.org/3/library/functions.html#float)

Arm - Enums

List of enums:

- **CalibrateMode**
- **RobotAxis**
- **JogShift**

class **CalibrateMode**(*value*)

Enumeration of Calibration Modes

AUTO= *1*

MANUAL= *2*

class **RobotAxis**(*value*)

Enumeration of Robot Axis : it used for Shift command

X= *0*

Y= *1*

Z= *2*

ROLL= *3*

PITCH= *4*

YAW= *5*

class **JogShift**(*value*)

Enumeration of Jog Shift : it used for Jog commands

JOINTS_SHIFT= *1*

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

POSE_SHIFT= *2*

Arm - Objects

- **HardwareStatusObject**

class **HardwareStatusObject**

Object used to store every hardware information

Variables:
- **rpi_temperature** (*float* (https://docs.python.org/3/library/functions.html#float)) – Number representing the rpi temperature
- **hardware_version** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – Number representing the hardware version
- **connection_up** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – Boolean indicating if the connection with the robot is up
- **error_message** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – Error message status on error
- **calibration_needed** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – Boolean indicating if a calibration is needed
- **calibration_in_progress** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – Boolean indicating if calibration is in progress
- **motor_names** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[str* (https://docs.python.org/3/library/stdtypes.html#str)*]*) – List of motor names
- **motor_types** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[str* (https://docs.python.org/3/library/stdtypes.html#str)*]*) – List of motor types
- **motors_temperature** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – List of motors_temperature
- **motors_voltage** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – List of motors_voltage
- **hardware_errors** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[int* (https://docs.python.org/3/library/functions.html#int)*]*) – List of hardware errors
- **hardware_error_messages** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[str* (https://docs.python.org/3/library/stdtypes.html#str)*]*) – List of hardware error messages

- **JointStateObject**

class **JointStateObject**

Object used to store every joint state information

Variables:
- **name** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[str* (https://docs.python.org/3/library/stdtypes.html#str)*]*) – List of joint names
- **position** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – List of joint positions
- **velocity** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – List of joint velocities
- **effort** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – List of joint efforts

- **PoseObject**

class **PoseObject**(*x*, *y*, *z*, *roll*, *pitch*, *yaw*)

Pose object which stores x, y, z, roll, pitch & yaw parameters

Variables:
- **x** (*float* (https://docs.python.org/3/library/functions.html#float)) – X (meter)
- **y** (*float* (https://docs.python.org/3/library/functions.html#float)) – Y (meter)
- **z** (*float* (https://docs.python.org/3/library/functions.html#float)) – Z (meter)
- **roll** (*float* (https://docs.python.org/3/library/functions.html#float)) – Roll (radian)
- **pitch** (*float* (https://docs.python.org/3/library/functions.html#float)) – Pitch (radian)
- **yaw** (*float* (https://docs.python.org/3/library/functions.html#float)) – Yaw (radian)

Tool

This file presents the different Tool - Command Functions, Tool - Enums & Tool - Niryo Topics available with the Tool API

Tool - Command Functions

This section reference all existing functions to control your robot, which include

- Using tools
- Using grippers
- Using the vacuum pump
- Using the electromagnet

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

- Management of the TCP

All functions to control the robot are accessible via an instance of the classNiryoRobot (index.html#niryorobot)

```
robot = NiryoRobot(<robot_ip_address>)

robot.tool.update_tool()
robot.tool.grasp_with_tool()
robot.tool.release_with_tool()
...
```

See examples on Examples Section (index.html#examples-tool-action)

List of functions subsections:

- Tool functions
- Grippers functions
- Vacuum pump functions
- Electromagnet functions
- TCP functions

class **Tool**(*client*)

Tool robot functions

Example:

```
ros_instance = NiryoRos("10.10.10.10") # Hotspot
tool_interface = Tool(ros_instance)
```

Parameters:
client (*NiryoRos* (index.html#pyniryo2.niryo_ros.NiryoRos)) – Niryo ROS client

Tool functions

Tool.update_tool(*callback=None, errback=None, timeout=None*)

Update equipped tool

Examples:

```
# Synchronous use
tool.update_tool()

# Asynchronous use
def update_tool_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Update failed")
    else:
        print("Update completed with success")

tool.update_tool(update_tool_callback)
```

Parameters:
- **callback** (*function*) – Callback invoked on successful execution.
- **errback** (*function*) – Callback invoked on error.
- **timeout** – Timeout for the operation, in seconds. Only used if blocking.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

property **Tool.tool**

Returns:
The equipped tool ID

Return type:
ToolID

property **Tool.get_current_tool_id**

Returns the equipped tool Id client which can be used synchronously or asynchronously to obtain the equipped tool Id. The topic returns a attribute of the ToolID enum.

Examples:

```
# Get last value
tool.get_current_tool_id()
tool.get_current_tool_id.value

# Subscribe a callback
def tool_id_callback(tool_id_object):
    print tool_id_object

tool.get_current_tool_id.subscribe(tool_id_callback)
tool.get_current_tool_id.unsubscribe()
```

Returns:
the equipped tool Id topic instance.

Return type:
NiryoTopic (index.html#pyniryo2.niryo_topic.NiryoTopic)

Tool.grasp_with_tool(*callback=None*)

Grasp with tool This action correspond to * Close gripper for Grippers * Pull Air for Vacuum pump * Activate for Electromagnet If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

Examples:

```
tool.update_tool()
tool.grasp_with_tool()

def tool_callback(_msg)
    print("Grasped")

tool.grasp_with_tool(callback=tool_callback)
```

Parameters:
callback (*function*) – Callback invoked on successful execution.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Tool.release_with_tool(*callback=None*)

Release with tool This action correspond to * Open gripper for Grippers * Push Air for Vacuum pump * Deactivate for Electromagnet If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

Examples:

```
tool.update_tool()
tool.release_with_tool()

def tool_callback(_msg)
    print("Released")

tool.release_with_tool(callback=tool_callback)
```

Parameters:
callback (*function*) – Callback invoked on successful execution.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Grippers functions

Tool.open_gripper(*speed=500, max_torque_percentage=100, hold_torque_percentage=30, callback=None*)

Open gripper associated to the equipped gripper. If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

Examples:

```
tool.update_tool()
tool.open_gripper()

# Niryo One and Ned
tool.open_gripper(speed=850)

# Ned2
tool.open_gripper(max_torque_percentage=100, hold_torque_percentage=50)

def tool_callback(_msg)
    print("Released")

tool.open_gripper(callback=tool_callback)
```

Parameters:

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

- **speed** (*int* (https://docs.python.org/3/library/functions.html#int)) – Between 100 & 1000 (only for Niryo One and Ned1)
- **max_torque_percentage** (*int* (https://docs.python.org/3/library/functions.html#int)) – Closing torque percentage (only for Ned2)
- **hold_torque_percentage** (*int* (https://docs.python.org/3/library/functions.html#int)) – Hold torque percentage after closing (only for Ned2)
- **callback** (*function*) – Callback invoked on successful execution.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Tool.close_gripper(*speed=500, max_torque_percentage=100, hold_torque_percentage=30, callback=None*)

Close gripper associated to 'gripper_id' with a speed 'speed' If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

Examples:

```
tool.update_tool()
tool.close_gripper()

# Niryo One and Ned
tool.close_gripper(speed=850)

# Ned2
tool.close_gripper(max_torque_percentage=100, hold_torque_percentage=50)

def tool_callback(_msg)
   print("Grasped")

tool.close_gripper(callback=tool_callback)
```

Parameters:

- **speed** (*int* (https://docs.python.org/3/library/functions.html#int)) – Between 100 & 1000 (only for Niryo One and Ned1)
- **max_torque_percentage** (*int* (https://docs.python.org/3/library/functions.html#int)) – Opening torque percentage (only for Ned2)
- **hold_torque_percentage** (*int* (https://docs.python.org/3/library/functions.html#int)) – Hold torque percentage after opening (only for Ned2)
- **callback** (*function*) – Callback invoked on successful execution.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Vacuum pump functions

Tool.pull_air_vacuum_pump(*callback=None*)

Pull air of vacuum pump If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

Examples:

```
tool.update_tool()
tool.pull_air_vacuum_pump()

def tool_callback(_msg)
   print("Grasped")

tool.pull_air_vacuum_pump(callback=tool_callback)
```

Parameters:
callback (*function*) – Callback invoked on successful execution.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Tool.push_air_vacuum_pump(*callback=None*)

Push air of vacuum pump If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

Examples:

```
tool.update_tool()
tool.push_air_vacuum_pump()

def tool_callback(_msg)
   print("Released")

tool.push_air_vacuum_pump(callback=tool_callback)
```

Parameters:
callback (*function*) – Callback invoked on successful execution.

Return type:

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

None (https://docs.python.org/3/library/constants.html#None)

Electromagnet functions

Tool.setup_electromagnet(*pin_id*)

Setup electromagnet on pin

Example:

```
tool.setup_electromagnet(PinID.GPIO_1A)
```

Parameters:
pin_id (*PinID*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Tool.activate_electromagnet(*pin_id=None*, *callback=None*)

Activate electromagnet associated to electromagnet_id on pin_id If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

Examples:

```
tool.setup_electromagnet(PinID.GPIO_1A)
tool.activate_electromagnet()
tool.activate_electromagnet(PinID.GPIO_1A)

def tool_callback(_msg)
   print("Grasped")

tool.activate_electromagnet(callback=tool_callback)
```

Parameters:
- **pin_id** (*PinID*) –
- **callback** (*function*) – Callback invoked on successful execution.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Tool.deactivate_electromagnet(*pin_id=None*, *callback=None*)

Deactivate electromagnet associated to electromagnet_id on pin_id If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

Examples:

```
tool.setup_electromagnet(PinID.GPIO_1A)
tool.deactivate_electromagnet()
tool.deactivate_electromagnet(PinID.GPIO_1A)

def tool_callback(_msg)
   print("Deactivated")

tool.deactivate_electromagnet(callback=tool_callback)
```

Parameters:
- **pin_id** (*PinID*) –
- **callback** (*function*) – Callback invoked on successful execution.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

TCP functions

Tool.enable_tcp(*enable=True*)

Enables or disables the TCP function (Tool Center Point). If activation is requested, the last recorded TCP value will be applied. The default value depends on the gripper equipped. If deactivation is requested, the TCP will be coincident with the tool_link.

Parameters:
enable (*Bool*) – True to enable, False otherwise.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Tool.set_tcp(**args*)

Activates the TCP function (Tool Center Point) and defines the transformation between the tool_link frame and the TCP frame.

Examples:

```
tool.set_tcp(0.02, 0.0, 0.03, 0.0, 1.57, 0.0)
tool.set_tcp([0.02, 0.0, 0.03, 0.0, 1.57, 0.0])
tool.set_tcp(PoseObject(0.02, 0.0, 0.03, 0.0, 1.57, 0.0))
```

Parameters:
args (*Union[tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#pyniryo2.objects.PoseObject)*]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a PoseObject

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Tool.reset_tcp()

Reset the TCP (Tool Center Point) transformation. The PCO will be reset according to the tool equipped.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Tool - Niryo Topics

The use of these functions is explained in the NiryoTopic (index.html#niryotopic) section. They allow the acquisition of data in real time by callbacks or by direct call.

Tool's Niryo Topics

Name	Function	Return type
/niryo_robot_tools_commander/current_id	get_current_tool_id	ToolID

Tool - Enums

List of enums:

- **ToolID**
- **ToolCommand**

class **ToolID**(*value*)

Enumeration of Tools IDs

NONE= *0*

GRIPPER_1= *11*

GRIPPER_2= *12*

GRIPPER_3= *13*

GRIPPER_4= *14*

ELECTROMAGNET_1= *30*

VACUUM_PUMP_1= *31*

class **ToolCommand**(*value*)

An enumeration.

OPEN_GRIPPER= *1*

CLOSE_GRIPPER= *2*

PULL_AIR_VACUUM_PUMP= *10*

PUSH_AIR_VACUUM_PUMP= *11*

SETUP_DIGITAL_IO= *20*

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

ACTIVATE_DIGITAL_IO= *21*

DEACTIVATE_DIGITAL_IO= *22*

Vision

This file presents the different Vision - Command functions, Vision - Enums, Vision - Niryo Topics & Vision - Namedtuple available with the Vision API

Vision - Command functions

This section reference all existing functions to control your robot arm, which include

- Getting camera image
- Detecting objects
- Managing workspaces

All functions to control the robot are accessible via an instance of the classNiryoRobot (index.html#niryorobot)

```
robot = NiryoRobot(<robot_ip_address>)

robot.vision.vision_pick("workspace_1", 0.0, ObjectShape.ANY, ObjectColor.ANY)
robot.vision.detect_object("workspace_1", ObjectShape.ANY, ObjectColor.ANY)
...
```

See examples on Examples Section (index.html#examples-vision)

List of functions subsections:

- Camera functions
- Detection functions
- Workspace functions

class **Vision**(*client*, *arm=None*, *tool=None*)

Vision robot functions

Example:

```
ros_instance = NiryoRos("10.10.10.10") # Hotspot
vision_interface = Vision(ros_instance)
```

Parameters:
client (*NiryoRos* (index.html#pyniryo2.niryo_ros.NiryoRos)) – Niryo ROS client

Camera functions

property **Vision.get_img_compressed**

Get image from video stream in a compressed format. Use uncompress_image from the vision package to uncompress it

Examples:

```
import pyniryo

img_compressed = vision.get_img_compressed()
camera_info = vision.get_camera_intrinsics()
img = pyniryo.uncompress_image(img_compressed)
img = pyniryo.undistort_image(img, camera_info.intrinsics, camera_info.distortion)
```

Returns:
string containing a JPEG compressed image

Return type:
NiryoTopic (index.html#pyniryo2.niryo_topic.NiryoTopic)

property **Vision.get_camera_intrinsics**

Get calibration object: camera intrinsics, distortions coefficients The topic return a namedtuple(intrinsics: list[list[float]], distortion: list[list[float]])

Examples:

```
vision.get_camera_intrinsics()
vision.get_camera_intrinsics().value

def camera_info_callback(camera_info):
    print(camera_info.intrinsics)
    print(camera_info.distortion)
    vision.get_camera_intrinsics.unsubscribe()

vision.get_camera_intrinsics.subscribe(camera_info_callback)
```

Return type:
NiryoTopic (index.html#pyniryo2.niryo_topic.NiryoTopic)

property **Vision.get_image_parameters**

Return the NiryoTopic to get last stream image parameters: Brightness factor, Contrast factor, Saturation factor. The topic return a namedtuple(brightness_factor: float, contrast_factor: float, saturation_factor: float)

Brightness factor: How much to adjust the brightness. 0.5 will give a darkened image, 1 will give the original image while 2 will enhance the brightness by a factor of 2.

Contrast factor: A factor of 1 gives original image. Making the factor towards 0 makes the image greyer, while factor>1 increases the contrast of the image.

Saturation factor: 0 will give a black and white image, 1 will give the original image while 2 will enhance the saturation by a factor of 2.

Examples:

```
vision.get_image_parameters()
vision.get_image_parameters.value

def image_parameters_callback(image_parameters):
    print(image_parameters.brightness_factor)
    print(image_parameters.contrast_factor)
    print(image_parameters.saturation_factor)

    vision.get_image_parameters.unsubscribe()

vision.get_image_parameters.subscribe(image_parameters_callback)
```

Returns:
ImageParameters namedtuple containing the brightness factor, contrast factor and saturation factor.

Return type:
NiryoTopic (index.html#pyniryo2.niryo_topic.NiryoTopic)

Vision.set_brightness(*brightness_factor*)

Modify image brightness

Parameters:
brightness_factor (*float* (https://docs.python.org/3/library/functions.html#float)) – How much to adjust the brightness. 0.5 will give a darkened image, 1 will give the original image while 2 will enhance the brightness by a factor of 2.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Vision.set_contrast(*contrast_factor*)

Modify image contrast

Parameters:
contrast_factor (*float* (https://docs.python.org/3/library/functions.html#float)) – A factor of 1 gives original image. Making the factor towards 0 makes the image greyer, while factor>1 increases the contrast of the image.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Vision.set_saturation(*saturation_factor*)

Modify image saturation

Parameters:
saturation_factor (*float* (https://docs.python.org/3/library/functions.html#float)) – How much to adjust the saturation. 0 will give a black and white image, 1 will give the original image while 2 will enhance the saturation by a factor of 2.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Detection functions

Vision.get_target_pose_from_cam(*workspace_name, height_offset=0.0, shape=<ObjectShape.ANY: 'ANY'>, color=<ObjectColor.ANY: 'ANY'>*)

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

First detects the specified object using the camera and then returns the robot pose in which the object can be picked with the current tool

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the workspace
- **height_offset** (*float* (https://docs.python.org/3/library/functions.html#float)) – offset between the workspace and the target height
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns:
object_found, object_pose, object_shape, object_color

Return type:
(bool (https://docs.python.org/3/library/functions.html#bool), PoseObject (index.html#pyniryo2.objects.PoseObject), ObjectShape, ObjectColor)

Vision.vision_pick(*workspace_name, height_offset=0.0, shape=<ObjectShape.ANY: 'ANY'>, color=<ObjectColor.ANY: 'ANY'>*)

Picks the specified object from the workspace. This function has multiple phases:

1. detect object using the camera
2. prepare the current tool for picking
3. approach the object
4. move down to the correct picking pose
5. actuate the current tool
6. lift the object

Example:

```
robot = NiryoRobot(ip_address="x.x.x.x")
robot.arm.calibrate_auto()
robot.arm.move_pose(<observation_pose>)
obj_found, shape_ret, color_ret = robot.vision.vision_pick(<workspace_name>,
                                        height_offset=0.0,
                                        shape=ObjectShape.ANY,
                                        color=ObjectColor.ANY)
```

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the workspace
- **height_offset** (*float* (https://docs.python.org/3/library/functions.html#float)) – offset between the workspace and the target height
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns:
object_found, object_shape, object_color

Return type:
(bool (https://docs.python.org/3/library/functions.html#bool), ObjectShape, ObjectColor)

Vision.move_to_object(*workspace_name, height_offset, shape, color*)

Same as *get_target_pose_from_cam* but directly moves to this position

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the workspace
- **height_offset** (*float* (https://docs.python.org/3/library/functions.html#float)) – offset between the workspace and the target height
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns:
object_found, object_shape, object_color

Return type:
(bool (https://docs.python.org/3/library/functions.html#bool), ObjectShape, ObjectColor)

Vision.detect_object(*workspace_name, shape=<ObjectShape.ANY: 'ANY'>, color=<ObjectColor.ANY: 'ANY'>*)

Detect object in workspace and return its pose and characteristics

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the workspace
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns:
object_found, object_pose, object_shape, object_color

Return type:

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

(bool (https://docs.python.org/3/library/functions.html#bool), PoseObject (index.html#pyniryo2.objects.PoseObject), str (https://docs.python.org/3/library/stdtypes.html#str), str (https://docs.python.org/3/library/stdtypes.html#str))

Workspace functions

Vision.get_target_pose_from_rel(*workspace_name, height_offset, x_rel, y_rel, yaw_rel*)

Given a pose (x_rel, y_rel, yaw_rel) relative to a workspace, this function returns the robot pose in which the current tool will be able to pick an object at this pose.

The height_offset argument (in m) defines how high the tool will hover over the workspace. If height_offset = 0, the tool will nearly touch the workspace.

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the workspace
- **height_offset** (*float* (https://docs.python.org/3/library/functions.html#float)) – offset between the workspace and the target height
- **x_rel** (*float* (https://docs.python.org/3/library/functions.html#float)) – x relative pose (between 0 and 1)
- **y_rel** (*float* (https://docs.python.org/3/library/functions.html#float)) – y relative pose (between 0 and 1)
- **yaw_rel** (*float* (https://docs.python.org/3/library/functions.html#float)) – Angle in radians

Returns:
target_pose

Return type:
PoseObject (index.html#pyniryo2.objects.PoseObject)

Vision.save_workspace_from_robot_poses(*workspace_name, pose_origin, pose_2, pose_3, pose_4*)

Save workspace by giving the poses of the robot to point its 4 corners with the calibration Tip. Corners should be in the good order. Markers' pose will be deduced from these poses

Poses should be either a list [x, y, z, roll, pitch, yaw] or a PoseObject

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – workspace name
- **pose_origin** (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#pyniryo2.objects.PoseObject)*]*) –
- **pose_2** (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#pyniryo2.objects.PoseObject)*]*) –
- **pose_3** (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#pyniryo2.objects.PoseObject)*]*) –
- **pose_4** (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#pyniryo2.objects.PoseObject)*]*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Vision.save_workspace_from_points(*workspace_name, point_origin, point_2, point_3, point_4*)

Save workspace by giving the points of worskpace's 4 corners. Points are written as [x, y, z] Corners should be in the good order.

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – workspace name
- **point_origin** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) –
- **point_2** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) –
- **point_3** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) –
- **point_4** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Vision.delete_workspace(*workspace_name*)

Delete workspace from robot's memory

Parameters:
workspace_name (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the saved workspace

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Vision.get_workspace_ratio(*workspace_name*)

Get workspace ratio from robot's memory

Parameters:
workspace_name (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – workspace name

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Return type:
float (https://docs.python.org/3/library/functions.html#float)

Vision.get_workspace_list()

Get list of workspaces' name store in robot's memory

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[str (https://docs.python.org/3/library/stdtypes.html#str)]

Vision - Niryo Topics

The use of these functions is explained in the NiryoTopic (index.html#niryotopic) section. They allow the acquisition of data in real time by callbacks or by direct call.

Vision's Niryo Topics

Name	Function	Return type
/niryo_robot_vision/compressed_video_stream	**get_img_compressed**	**list** (https://docs.python.org/3/library/stdtypes.html#list) [**numpy.uint8**]
/niryo_robot_vision/camera_intrinsics	**get_camera_intrinsics**	**CameraInfo**

Vision - Enums

List of enums:

- **ObjectColor**
- **ObjectShape**
- **ManageWorkspace**

class **ObjectColor**(*value*)

Enumeration of Colors available for image processing

RED= *'RED'*

BLUE= *'BLUE'*

GREEN= *'GREEN'*

ANY= *'ANY'*

class **ObjectShape**(*value*)

Enumeration of Shapes available for image processing

SQUARE= *'SQUARE'*

CIRCLE= *'CIRCLE'*

ANY= *'ANY'*

class **ManageWorkspace**(*value*)

Enumeration of actions available for workspaces management

SAVE= *1*

SAVE_WITH_POINTS= *2*

DELETE= *-1*

Vision - Namedtuple

class **CameraInfo**(*intrinsics, distortion*)

Create new instance of CameraInfo(intrinsics, distortion)

distortion

Alias for field number 1

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

intrinsics

Alias for field number 0

class **ImageParameters**(*brightness_factor*, *contrast_factor*, *saturation_factor*)

Create new instance of ImageParameters(brightness_factor, contrast_factor, saturation_factor)

brightness_factor

Alias for field number 0

contrast_factor

Alias for field number 1

saturation_factor

Alias for field number 2

I/Os

This file presents the different I/Os - Command functions, I/Os - Enums, I/Os - Niryo Topics & I/Os - Objects available with the Arm API

I/Os - Command functions

check_ned2_version(*func*)

Decorator that check the robot version

check_ned_one_version(*func*)

Decorator that check the robot version

This section reference all existing functions to control your robot, which include

- Getting IOs status
- Setting IOs mode
- Setting IOs value

All functions to control the robot are accessible via an instance of the classNiryoRobot (index.html#niryorobot)

```
robot = NiryoRobot(<robot_ip_address>)

robot.io.set_pin_mode(PinID.GPIO_1A, PinMode.INPUT)
robot.io.digital_write(PinID.GPIO_1A, PinState.HIGH)
...
```

See examples on Examples Section (index.html#examples-conveyor)

List of functions subsections:

- State functions
- Read & Write functions

class **IO**(*client*)

IO robot functions

Example:

```
ros_instance = NiryoRos("10.10.10.10") # Hotspot
io_interface = IO(ros_instance)
```

Parameters:
client (*NiryoRos* (index.html#pyniryo2.niryo_ros.NiryoRos)) – Niryo ROS client

State functions

property **IO.digital_io_states**

Return the value of digital ios.

Returns:
State, Name, Mode of the pin

Return type:

DigitalPinObject

property **IO.get_digital_io_states**

Returns the io state client which can be used synchronously or asynchronously to obtain the io states. The io state client returns a list of DigitalPinObject.

Examples:

```
# Get last value
io.get_digital_io_states()
io.get_digital_io_states.value

# Subscribe a callback
def io_callback(io_state):
    print io_state

io.get_digital_io_states.subscribe(io_callback)
io.get_digital_io_states.unsubscribe()
```

Returns:
io state topic instance

Return type:
NiryoTopic (index.html#pyniryo2.niryo_topic.NiryoTopic)

IO.get_digital_io_state(*pin_id*)

Return the value of a digital io.

Returns:
digital io object

Return type:
DigitalPinObject

Read & Write functions

IO.set_pin_mode(***kwargs*)

IO.digital_write(*pin_id, digital_state*)

Set pin_id state to digital_state

Examples:

```
io.digital_write(PinID.GPIO_1A, PinState.HIGH)
io.digital_write('1A', PinState.LOW)
```

Parameters:

- **pin_id** (*PinID or String*) –
- **digital_state** (*PinState or bool* (https://docs.python.org/3/library/functions.html#bool)) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

IO.digital_read(*pin_id*)

Return the value of a digital pin.

Examples:

```
io.set_pin_mode(PinID.GPIO_1A, PinMode.OUTPUT)
io.digital_read(PinID.GPIO_1A)
```

Parameters:
pin_id (*PinID or str* (https://docs.python.org/3/library/stdtypes.html#str)) –

Return type:
bool (https://docs.python.org/3/library/functions.html#bool)

I/Os - Niryo Topics

The use of these functions is explained in the NiryoTopic (index.html#niryotopic) section. They allow the acquisition of data in real time by callbacks or by direct call.

I/O's Niryo Topics

Name	Function	Return type
/niryo_robot_rpi/digital_io_state	**get_digital_io_states**	**list** (https://docs.python.org/3/library/stdtypes.html#list) [**DigitalPinObject**]

I/Os - Enums

List of enums:

- **PinMode**
- **PinState**
- **PinID**

class **PinMode**(*value*)

Enumeration of Pin Modes

OUTPUT= *0*

INPUT= *1*

class **PinState**(*value*)

Pin State is either LOW or HIGH

LOW= *False*

HIGH= *True*

class **PinID**(*value*)

Enumeration of Robot Pins

GPIO_1A= *'1A'*

GPIO_1B= *'1B'*

GPIO_1C= *'1C'*

GPIO_2A= *'2A'*

GPIO_2B= *'2B'*

GPIO_2C= *'2C'*

SW_1= *'SW1'*

SW_2= *'SW2'*

DO1= *'DO1'*

DO2= *'DO2'*

DO3= *'DO3'*

DO4= *'DO4'*

DI1= *'DI1'*

DI2= *'DI2'*

DI3= *'DI3'*

DI4= *'DI4'*

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

DI5= *'DI5'*

AI1= *'AI1'*

AI2= *'AI2'*

AO1= *'AO1'*

AO2= *'AO2'*

I/Os - Objects

class **DigitalPinObject**(*name, mode, value*)

Object used to store information on digital pins

class **AnalogPinObject**(*name, mode, value*)

Object used to store information on digital pins

Conveyor

This file presents the different Conveyor - Command functions, Conveyor - Enums, Conveyor - Niryo Topics & Conveyor - Namedtuple available with the Arm API

Conveyor - Command functions

This section reference all existing functions to control your robot, which include

- Controlling conveyors

All functions to control the robot are accessible via an instance of the classNiryoRobot (index.html#niryorobot)

```
robot = NiryoRobot(<robot_ip_address>)

conveyor_id = robot.conveyor.set_conveyor()
robot.conveyor.run_conveyor(conveyor_id)
...
```

See examples on Examples Section (index.html#examples-conveyor)

List of functions subsections:

- Conveyor functions

Conveyor functions

class **Conveyor**(*client*)

Conveyor robot functions

Example:

```
ros_instance = NiryoRos("10.10.10.10") # Hotspot
conveyor_interface = Conveyor(ros_instance)
```

Parameters:
client (*NiryoRos* (index.html#pyniryo2.niryo_ros.NiryoRos)) – Niryo ROS client

set_conveyor()

Scan if a conveyor is plugged or not on a can bus. If a new conveyor is detected, activate it and return its conveyor ID. If a conveyor is already set, return its ID

Example:

```
# Get the id of the conveyor plugged
conveyor_id = conveyor.set_conveyor()

# Scan and set the conveyor plugged
conveyor.set_conveyor()
```

:return : New conveyor ID :rtype: ConveyorID

unset_conveyor(*conveyor_id*)

Remove and unset a conveyor previously plugged and set

Example:

```
conveyor_id = conveyor.set_conveyor()
conveyor.unset_conveyor(conveyor_id)
conveyor.unset_conveyor(ConveyorID.ID_1)
conveyor.unset_conveyor(ConveyorID.ID_2)
```

Parameters:
conveyor_id (*ConveyorID*) – Basically, ConveyorID.ID_1 or ConveyorID.ID_2

Return type:
None (https://docs.python.org/3/library/constants.html#None)

run_conveyor(*conveyor_id*, *speed=100*, *direction=<ConveyorDirection.FORWARD: 1>*)

Run conveyor at id 'conveyor_id'

Example:

```
# Set the conveyor and get its id and un it.
# By default, the conveyor will go forward at a speed of 50
# You can't choose the parameters with this method

conveyor_id = conveyor.set_conveyor()
conveyor.run_conveyor(conveyor_id)
```

Parameters:

- **conveyor_id** (*ConveyorID*) – The conveyor id
- **speed** (*int* (https://docs.python.org/3/library/functions.html#int)) – speed percentage between 0% and 100%
- **direction** (*ConveyorDirection*) – direction = ConveyorDirection.FORWARD

Return type:
None (https://docs.python.org/3/library/constants.html#None)

stop_conveyor(*conveyor_id*)

Stop conveyor at id 'conveyor_id'

Example:

```
# Set the conveyor and get its id, run it and then stop it after 3 seconds
# By default, the conveyor will go forward at a speed of 50
# When the conveyor is stopped, its control_on parameter is False and its speed is 0

import time

conveyor_id = conveyor.set_conveyor()
conveyor.run_conveyor(conveyor_id)
time.sleep(3)
conveyor.stop_conveyor(conveyor_id)
```

Parameters:
conveyor_id (*ConveyorID*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

control_conveyor(*conveyor_id*, *control_on*, *speed*, *direction*)

Control conveyor associated to conveyor_id. Then stops it if bool_control_on is False, else refreshes it speed and direction

Example:

```
# Example 1
# Set the conveyor and get its id, control it and then stop it after 3 seconds
# It this first example, we control the conveyor at a speed of 100% and in the forward direction

import time

conveyor_id = conveyor.set_conveyor()
conveyor.control_conveyo(conveyor_id, True, 100, ConveyorDirection.FORWARD.value)
time.sleep(3)
conveyor.stop_conveyor(conveyor_id)
```

# Example 2

Set the conveyor and get its id, control it and then stop it after 3 seconds # It this second example, we control the conveyor at a speed of 30% and in the backward direction

import time

conveyor_id = conveyor.set_conveyor() conveyor.control_conveyor(conveyor_id, True, 30, ConveyorDirection.BACKWARD.value) time.sleep(3) conveyor.stop_conveyor(conveyor_id)

Parameters:

- **conveyor_id** (*ConveyorID*) –
- **control_on** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – True for activate, False for deactivate
- **speed** (*int* (https://docs.python.org/3/library/functions.html#int)) – New speed which is a percentage of maximum speed (0% to 100%)
- **direction** (*ConveyorDirection*) – ConveyorDirection.FORWARD.value, ConveyorDirection.BACKWARD.value

Return type:
None (https://docs.python.org/3/library/constants.html#None)

property **get_conveyors_feedback**

Returns the conveyors feedback client which can be used synchronously or asynchronously to obtain the conveyors feedback: (conveyor_id, connection_state, running, speed, direction)

Examples:

```
# Get last value
arm.get_conveyors_feedback()
arm.get_conveyors_feedback.value

# Subscribe a callback
def conveyor_callback(conveyor_feedback):
    print conveyor_feedback

arm.hardware_status.subscribe(conveyor_callback)
# wait
arm.hardware_status.unsubscribe()
```

Returns:
namedtuple[conveyor_id, running, speed, direction]

Return type:
namedtuple(ConveyorID, bool (https://docs.python.org/3/library/functions.html#bool), int (https://docs.python.org/3/library/functions.html#int), ConveyorDirection)

property **conveyors**

Return list of registered conveyors

Returns:
namedtuple[conveyor_id, running, speed, direction]

Return type:
namedtuple(ConveyorID, bool (https://docs.python.org/3/library/functions.html#bool), int (https://docs.python.org/3/library/functions.html#int), ConveyorDirection)

Conveyor - Niryo Topics

The use of these functions is explained in the NiryoTopic (index.html#niryotopic) section. They allow the acquisition of data in real time by callbacks or by direct call.

Conveyors's Niryo Topics

Name	Function	Return type
/niryo_robot/conveyor/feedback	**get_conveyors_feedback**	**list** (https://docs.python.org/3/library/stdtypes.html#list) [**ConveyorInfo**]

Conveyor - Enums

List of enums:

- **ConveyorID**
- **ConveyorDirection**
- **ConveyorStatus**

class **ConveyorID**(*value*)

ConveyorID to be able to have CAN (id 12 and 13) and TTL (id 9 and 10) conveyor in any possible combination

ID_1 = 12 # One, Ned ID_2 = 13 # One, Ned ID_3 = 9 # Ned2 ID_4 = 10 # Ned2

NONE= *0*

ID_1= *-1*

ID_2= *-2*

class **ConveyorCan**(*value*)

ConveyorID to control conveyors with CAN interface

NONE= *0*

ID_1= *12*

ID_2= *13*

class **ConveyorTTL**(*value*)

ConveyorID to control conveyors with TTL interface

NONE= *0*

ID_1= *9*

ID_2= *10*

class **ConveyorDirection**(*value*)

Enumeration of the directions of the conveyor

FORWARD= *1*

BACKWARD= *-1*

class **ConveyorStatus**(*value*)

Enumeration of the different Conveyor status

ADD= *1*

REMOVE= *2*

Conveyor - Namedtuple

class **ConveyorInfo**(*conveyor_id, running, speed, direction*)

Create new instance of ConveyorInfo(conveyor_id, running, speed, direction)

conveyor_id

Alias for field number 0

direction

Alias for field number 3

running

Alias for field number 1

speed

Alias for field number 2

Saved poses

This file presents the different Saved poses - Command functions, available with the Saved poses API

Saved poses - Command functions

This section reference all existing functions to control your robot, which include

- Management of saved poses

All functions to control the robot are accessible via an instance of the classNiryoRobot (index.html#niryorobot)

```
robot = NiryoRobot(<robot_ip_address>)

pose_name_list = robot.saved_poses.get_saved_pose_list()
robot.saved_poses.get_pose_saved(pose_name_list[0])
...
```

See examples on Examples Section (index.html#examples-conveyor)

List of functions subsections:

- Saved poses functions

Saved poses functions

class **SavedPoses**(*client*)

get_pose_saved(*pose_name*)

Get pose saved in from Ned's memory

Examples:

```
pose = saved_poses.get_pose_saved("pose1")
```

Parameters:
pose_name (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – Pose name in robot's memory

Returns:
Pose associated to pose_name

Return type:
PoseObject (index.html#pyniryo2.objects.PoseObject)

save_pose(*pose_name*, **args*)

Save pose (x, y, z, roll, pitch, yaw) in robot's memory

Examples:

```
saved_poses.save_pose("pose1", 0.3, 0.0, 0.3, 0.0, 1.57, 0.0)
saved_poses.save_pose("pose1", [0.3, 0.0, 0.3, 0.0, 1.57, 0.0])
saved_poses.save_pose("pose1", PoseObject(0.3, 0.0, 0.3, 0.0, 1.57, 0.0))
```

Parameters:
args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#pyniryo2.objects.PoseObject)*]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a PoseObject

Return type:
None (https://docs.python.org/3/library/constants.html#None)

delete_pose(*pose_name*)

Delete pose from robot's memory

Examples:

```
if "pose1" in saved_poses.get_saved_pose_list():
   saved_poses.delete_pose("pose1")
```

Return type:
None (https://docs.python.org/3/library/constants.html#None)

get_saved_pose_list()

Get list of poses' name saved in robot memory

Examples:

```
>> print(saved_poses.get_saved_pose_list())
["pose1", "pose2", "pose3"]
```

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[str (https://docs.python.org/3/library/stdtypes.html#str)]

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Pick & Place

This file presents the different Pick & Place - Command functions available with the Pick & Place API

Pick & Place - Command functions

This section reference all existing functions to control your robot, which include

- Picking objects
- Placing objects

All functions to control the robot are accessible via an instance of the classNiryoRobot (index.html#niryorobot)

```
robot = NiryoRobot(<robot_ip_address>)

robot.pick_place.pick_from_pose([0.2, 0.0, 0.1, 0.0, 1.57, 0.0])
robot.pick_place.place_from_pose([0.0, 0.2, 0.1, 0.0, 1.57, 0.0])
...
```

See examples on Examples Section (index.html#examples-vision)

List of functions subsections:

- Pick & Place functions

Pick & Place functions

class **PickPlace**(*client*, *arm=None*, *tool=None*, *trajectories=None*)

pick_from_pose(**args*)

Execute a picking from a pose.

A picking is described as :

* going over the object
* going down until height = z
* grasping with tool
* going back over the object

Parameters:
args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#pyniryo2.objects.PoseObject)*]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a PoseObject

Return type:
None (https://docs.python.org/3/library/constants.html#None)

place_from_pose(**args*)

Execute a placing from a position.

A placing is described as :

* going over the place
* going down until height = z
* releasing the object with tool
* going back over the place

Parameters:
args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#pyniryo2.objects.PoseObject)*]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a PoseObject

Return type:
None (https://docs.python.org/3/library/constants.html#None)

pick_and_place(*pick_pose*, *place_pose*, *dist_smoothing=0.0*)

Execute a pick then a place

Parameters:

- **pick_pose** (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#pyniryo2.objects.PoseObject)*]*) – Pick Pose : [x, y, z, roll, pitch, yaw] or PoseObject

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

- **place_pose** (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#pyniryo2.objects.PoseObject)*]*) – Place Pose : [x, y, z, roll, pitch, yaw] or PoseObject
- **dist_smoothing** (*float* (https://docs.python.org/3/library/functions.html#float)) – Distance from waypoints before smoothing trajectory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Trajectories

This file presents the different Trajectories - Command functions available with the Trajectories API

Trajectories - Command functions

This section reference all existing functions to control your robot, which include

- Playing smoothed waypointed trajectories
- Managing saved trajectories

All functions to control the robot are accessible via an instance of the classNiryoRobot (index.html#niryorobot)

```
robot = NiryoRobot(<robot_ip_address>)

trajectories = robot.trajectories.get_saved_trajectory_list()
if len(trajectories) > 0:
   robot.trajectories.execute_trajectory_saved(trajectories[0])
...
```

See examples on Examples Section (index.html#examples-movement)

List of functions subsections:

- Trajectories functions

Trajectories functions

class **Trajectories**(*client, action_timeout=3600*)

get_saved_trajectory(*trajectory_name*)

Get saved trajectory from robot intern storage Will raise error if position does not exist

Example:

```
trajectories.get_saved_trajectory("trajectory_01")
```

Parameters:
trajectory_name (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the trajectory

Raises:
NiryoRosWrapperException – If trajectory file doesn't exist

Returns:
list of [j1, j2, j3, j4, j5, j6] in rad

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]]

execute_registered_trajectory(*trajectory_name*, *callback=None*)

Execute trajectory from Ned's memory If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

Examples:

```
trajectories.execute_trajectory_saved("trajectory_01")

from threading import Event
trajectory_event = Event()
trajectory_event.clear()

def trajectory_callback(result):
    print(result)
    trajectory_event.set()

trajectories.execute_trajectory_saved("trajectory_01", callback=trajectory_callback)
trajectory_event.wait()
```

Parameters:
callback (*function*) – Callback invoked on successful execution.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

execute_trajectory_from_poses(*list_poses*, *dist_smoothing=0.0*, *callback=None*)

Execute trajectory from list of poses If a callback function is not passed in parameter, the function will be blocking. Otherwise, the callback will be called when the execution of the function is finished.

Examples:

```
trajectory = [[0.3, 0.1, 0.3, 0., 0., 0., 1.],
              [0.3, -0.1, 0.3, 0., 0., 0., 1.],
              [0.3, -0.1, 0.4, 0., 0., 0., 1.],
              [0.3, 0.1, 0.4, 0., 0., 0., 1.]]

trajectories.execute_trajectory_from_poses(trajectory)
trajectories.execute_trajectory_from_poses(trajectory, dist_smoothing=0.02)
trajectories.execute_trajectory_from_poses([[0.3, 0.1, 0.3, 0., 0., 0., 1.], #[x,y,z,qx,qy,qz,qw]
                                            PoseObject(0.3, -0.1, 0.3, 0., 0., 0.),
                                            [0.3, -0.1, 0.4, 0., 0., 0.], #[x,y,z,roll,pitch,yaw]
                                            PoseObject(0.3, 0.1, 0.4, 0., 0., 0.)])

from threading import Event
trajectory_event = Event()
trajectory_event.clear()

def trajectory_callback(result):
    print(result)
    trajectory_event.set()

trajectories.execute_trajectory_from_poses(trajectory, callback=trajectory_callback)
trajectory_event.wait()
```

Parameters:

- **callback** (*function*) – Callback invoked on successful execution.
- **list_poses** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[Union[tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#pyniryo2.objects.PoseObject)*]]*) – List of: [x,y,z,qx,qy,qz,qw] or [x,y,z,roll,pitch,yaw] or PoseObject
- **dist_smoothing** (*float* (https://docs.python.org/3/library/functions.html#float)) – Distance from waypoints before smoothing trajectory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

save_trajectory(*trajectory*, *trajectory_name*, *description*)

Save trajectory in robot's memory

Examples:

```
trajectory = [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
              [1.57, 0.0, 0.0, 0.0, -1.57, 0.0],
              [-1.57, 0.0, 0.0, 0.0, -1.57, 0.0]]

trajectories.save_trajectory(trajectory, "trajectory_1", "test description trajectory_1")
```

Parameters:

- **trajectory** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[joints]*) – list of joints positions the robot needs to pass by in the trajectory
- **trajectory_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name you give trajectory in the robot's memory
- **description** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – description you give trajectory in the robot's memory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

save_last_learned_trajectory(*trajectory_name*, *description*)

Save last executed trajectory in robot's memory

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Examples:

```
trajectories.save_last_learned_trajectory("trajectory_1", "test description trajectory_1")
```

Parameters:

- **trajectory_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name you give trajectory in the robot's memory
- **description** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – description you give trajectory in the robot's memory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

delete_trajectory(*trajectory_name*)

Delete trajectory from robot's memory

Example:

```
if "trajectory_1" in trajectories.get_saved_trajectory_list():
    trajectories.delete_trajectory("trajectory_1")
```

Return type:
None (https://docs.python.org/3/library/constants.html#None)

clean_trajectory_memory()

Delete all trajectories from robot's memory

Example:

```
trajectories.clean_trajectory_memory()
```

Return type:
None (https://docs.python.org/3/library/constants.html#None)

get_saved_trajectory_list()

Get list of trajectories' name saved in robot memory

Example:

```
if "trajectory_1" in trajectories.get_saved_trajectory_list():
    trajectories.delete_trajectory("trajectory_1")
```

Returns:
list of tuple(trajectory name, trajectory definition)

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[tuple (https://docs.python.org/3/library/stdtypes.html#tuple)(str (https://docs.python.org/3/library/stdtypes.html#str), str (https://docs.python.org/3/library/stdtypes.html#str))]

update_trajectory_infos(*name*, *new_name*, *description=''*)

Update the trajectory informations: name and description

Example:

```
trajectories.update_trajectory_infos("trajectory_1", "trajectory_2", callback="change description")
```

Parameters:

- **name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the trajectory you want to change infos
- **new_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – new name you want to give to the trajectory
- **description** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – new description you want to give to the trajectory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Frames

This file presents the different Frames - Command functions available with the Frames API

All functions to control the robot are accessible via an instance of the classNiryoRobot (index.html#niryorobot)

```
robot = NiryoRobot(<robot_ip_address>)

frames = frames.get_saved_dynamic_frame_list()
...
```

Frames - Command functions

List of functions subsections:

- Frames functions

Frames functions

class **Frames**(*client*)

get_saved_dynamic_frame_list()

Get list of saved dynamic frames

Example:

```
list_frame, list_desc = robot.frames.get_saved_dynamic_frame_list()
print(list_frame)
print(list_desc)
```

Returns:
list of dynamic frames name, list of description of dynamic frames

Return type:
dictionnaire{name:description}

get_saved_dynamic_frame(*frame_name*)

Get name, description and pose of a dynamic frame

Example:

```
frame = robot.frames.get_saved_dynamic_frame("default_frame")
```

Parameters:
frame_name (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the frame

Returns:
name, description, position and orientation of a frame

Return type:
namedtuple(name(str (https://docs.python.org/3/library/stdtypes.html#str)), description(str (https://docs.python.org/3/library/stdtypes.html#str)), position(list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]), orientation(list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]))

save_dynamic_frame_from_poses(*frame_name*, *description*, *pose_origin*, *pose_x*, *pose_y*, *belong_to_workspace=False*)

Create a dynamic frame with 3 poses (origin, x, y)

Example:

```
pose_o = [0.1, 0.1, 0.1, 0, 0, 0]
pose_x = [0.2, 0.1, 0.1, 0, 0, 0]
pose_y = [0.1, 0.2, 0.1, 0, 0, 0]

robot.frames.save_dynamic_frame_from_poses("name", "une description test", pose_o, pose_x, pose_y)
```

Parameters:

- **frame_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the frame
- **description** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – description of the frame
- **pose_origin** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*] [x, y, z, roll, pitch, yaw]*) – pose of the origin of the frame
- **pose_x** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*] [x, y, z, roll, pitch, yaw]*) – pose of the point x of the frame
- **pose_y** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*] [x, y, z, roll, pitch, yaw]*) – pose of the point y of the frame
- **belong_to_workspace** (*boolean*) – indicate if the frame belong to a workspace

Returns:
status, message

Return type:

(int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

save_dynamic_frame_from_points(*frame_name*, *description*, *point_origin*, *point_x*, *point_y*, *belong_to_workspace=False*)

Create a dynamic frame with 3 points (origin, x, y)

Example:

```
point_o = [-0.1, -0.1, 0.1]
point_x = [-0.2, -0.1, 0.1]
point_y = [-0.1, -0.2, 0.1]

robot.frames.save_dynamic_frame_from_points("name", "une description test", point_o, point_x, point_y)
```

Parameters:

- **frame_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the frame
- **description** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – description of the frame
- **point_origin** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*] [x, y, z]*) – origin point of the frame
- **point_x** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*] [x, y, z]*) – point x of the frame
- **point_y** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*] [x, y, z]*) – point y of the frame
- **belong_to_workspace** (*boolean*) – indicate if the frame belong to a workspace

Returns:

status, message

Return type:

(int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

edit_dynamic_frame(*frame_name*, *new_frame_name*, *new_description*)

Modify a dynamic frame

Example:

```
robot.frames.edit_dynamic_frame("name", "new_name", "new description")
```

Parameters:

- **frame_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the frame
- **new_frame_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – new name of the frame
- **new_description** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – new description of the frame

Returns:

status, message

Return type:

(int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

delete_saved_dynamic_frame(*frame_name*, *belong_to_workspace=False*)

Delete a dynamic frame

Example:

```
robot.frames.delete_saved_dynamic_frame("name")
```

Parameters:

- **frame_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the frame to remove
- **belong_to_workspace** (*boolean*) – indicate if the frame belong to a workspace

Returns:

status, message

Return type:

(int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

Led Ring

This file presents the different functions, enums, topics and objects available with the Led Ring API

Led Ring - Command functions

This section reference all existing functions to control the Led Ring, which are several parameterizable animations. All functions are accessible via an instance of the class NiryoRobot (index.html#niryorobot)

```
robot = NiryoRobot(<robot_ip_address>)

robot.led_ring.solid([255, 255, 255])
...
```

List of functions:

- **LedRing**

class **LedRing**(*client*)

LedRing robot functions

Example:

```
ros_instance = NiryoRos("10.10.10.10") # Hotspot
led_ring_interface = LedRing(ros_instance)
```

Parameters:
client (*NiryoRos* (index.html#pyniryo2.niryo_ros.NiryoRos)) – Niryo ROS client

status

Returns the Led Ring status client which can be used synchronously or asynchronously to obtain the current Led Ring status (cf LedRingStatusObject).

Examples:

```
# Get last value
led_ring.status()
led_ring.status.value

# Subscribe a callback
def status_callback(msg):
    print([msg.r, msg.g, msg.b])

led_ring.status.subscribe(status_callback)
led_ring.status.unsubscribe()
```

Returns:
Led Ring status topic.

Return type:
NiryoTopic (index.html#pyniryo2.niryo_topic.NiryoTopic)

get_status()

Get Led Ring status.

Example:

```
status = led_ring.get_status()
print(status.animation)
```

Returns:
Object with the current led ring mode, the animation played and the color used

Return type:
LedRingStatusObject (index.html#pyniryo2.led_ring.objects.LedRingStatusObject)

solid(*color*)

Set the whole Led Ring to a fixed color.

Example:

```
led_ring.solid([15, 50, 255])
```

Parameters:
color (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

turn_off()

Turn off all LEDs

Example:

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
led_ring.turn_off()
```

Return type:
None (https://docs.python.org/3/library/constants.html#None)

flash(*color*, *period=0*, *iterations=0*, *wait=False*, *callback=None*, *timeout=None*)

Flashes a color according to a frequency. The frequency is equal to 1 / period.

Examples:

```
# Synchronous use
led_ring.flash([15, 50, 255])  # Non-blocking
led_ring.flash([15, 50, 255], 1, 100, False)  # Non-blocking
led_ring.flash([15, 50, 255], iterations=20, wait=True)  # Wait the end

frequency = 20  # Hz
total_duration = 10 # seconds
led_ring.flash([15, 50, 255], 1./frequency, total_duration * frequency , True)

# Asynchronous use
def led_ring_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Failed")
    else:
        print("Completed with success")

led_ring.flash([15, 50, 255], iterations=20, wait=True, callback=calibration_callback)
```

Parameters:

- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive flashes. If 0, the Led Ring flashes endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – The service wait for the animation to finish all iterations or not to answer. If iterations is 0, the service answers immediately.
- **callback** (*function*) – Callback invoked on successful execution.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

alternate(*color_list*, *period=0*, *iterations=0*, *wait=False*, *callback=None*, *timeout=None*)

Several colors are alternated one after the other.

Examples:

```
# Synchronous use
color_list = [
    [15, 50, 255],
    [255, 0, 0],
    [0, 255, 0],
]

led_ring.alternate(color_list) # Non-blocking
led_ring.alternate(color_list, 1, 100, False) # Non-blocking
led_ring.alternate(color_list, iterations=20, wait=True) # Blocking

# Asynchronous use
def led_ring_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Failed")
    else:
        print("Completed with success")

led_ring.alternate(color_list, iterations=20, wait=True, callback=calibration_callback)
```

Parameters:

- **color_list** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – Led color list of lists of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive alternations. If 0, the Led Ring alternates endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – The service wait for the animation to finish all iterations or not to answer. If iterations is 0, the service answers immediately.
- **callback** (*function*) – Callback invoked on successful execution.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

chase(*color*, *period=0*, *iterations=0*, *wait=False*, *callback=None*, *timeout=None*)

Movie theater light style chaser animation.

Examples:

```
# Synchronous use
led_ring.chase([15, 50, 255]) # Non-blocking
led_ring.chase([15, 50, 255], 1, 100, False) # Non-blocking
led_ring.chase([15, 50, 255], iterations=20, wait=True) # Blocking

# Asynchronous use
def led_ring_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Failed")
    else:
        print("Completed with success")

led_ring.chase([15, 50, 255], iterations=20, wait=True, callback=calibration_callback)
```

Parameters:

- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive chase. If 0, the animation continues endlessly. One chase just lights one Led every 3 LEDs.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – The service wait for the animation to finish all iterations or not to answer. If iterations is 0, the service answers immediately.
- **callback** (*function*) – Callback invoked on successful execution.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

wipe(*color*, *period=0*, *wait=False*, *callback=None*, *timeout=None*)

Wipe a color across the Led Ring, light a Led at a time.

Examples:

```
# Synchronous use
robot.wipe([15, 50, 255])  # Non-blocking
led_ring.wipe([15, 50, 255], 1, False)  # Non-blocking
led_ring.wipe([15, 50, 255], wait=True)  # Blocking

# Asynchronous use
def led_ring_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Failed")
    else:
        print("Completed with success")

led_ring.wipe([15, 50, 255], wait=True, callback=calibration_callback)
```

Parameters:

- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – The service wait for the animation to finish or not to answer.
- **callback** (*function*) – Callback invoked on successful execution.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

go_up(*color*, *period=0*, *iterations=0*, *wait=False*, *callback=None*, *timeout=None*)

LEDs turn on like a loading circle, and are then all turned off at once.

Examples:

```
# Synchronous use
led_ring.go_up([15, 50, 255])  # Non-blocking
led_ring.go_up([15, 50, 255], 1, 100, False)  # Non-blocking
led_ring.go_up([15, 50, 255], iterations=20, wait=True)  # Blocking

# Asynchronous use
def led_ring_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Failed")
    else:
        print("Completed with success")

led_ring.go_up([15, 50, 255], period=2, iterations=20, wait=True, callback=calibration_callback)
```

Parameters:

- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive turns around the Led Ring. If 0, the animation continues endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – The service wait for the animation to finish or not to answer. If iterations is 0, the service answers immediately.
- **callback** (*function*) – Callback invoked on successful execution.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:

None (https://docs.python.org/3/library/constants.html#None)

go_up_down(*color*, *period=0*, *iterations=0*, *wait=False*, *callback=None*, *timeout=None*)

LEDs turn on like a loading circle, and are turned off the same way.

Examples:

```
# Synchronous use
led_ring.go_up_down([15, 50, 255])  # Non-blocking
led_ring.go_up_down([15, 50, 255], 1, 100, False)  # Non-blocking
led_ring.go_up_down([15, 50, 255], iterations=20, wait=True)  # Blocking

# Asynchronous use
def led_ring_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Failed")
    else:
        print("Completed with success")

led_ring.go_up_down([15, 50, 255], period=2, iterations=20, wait=True, callback=calibration_callback)
```

Parameters:

- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive turns around the Led Ring. If 0, the animation continues endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – The service wait for the animation to finish or not to answer. If iterations is 0, the service answers immediately.
- **callback** (*function*) – Callback invoked on successful execution.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:

None (https://docs.python.org/3/library/constants.html#None)

breath(*color*, *period=0*, *iterations=0*, *wait=False*, *callback=None*, *timeout=None*)

Variation of the light intensity of the LED ring, similar to human breathing.

Examples:

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
# Synchronous use
led_ring.breath([15, 50, 255])  # Non-blocking
led_ring.breath([15, 50, 255], 1, 100, False)  # Non-blocking
led_ring.breath([15, 50, 255], iterations=20, wait=True)  # Blocking

# Asynchronous use
def led_ring_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Failed")
    else:
        print("Completed with success")

led_ring.breath([15, 50, 255], period=2, iterations=20, wait=True,
    callback=calibration_callback)
```

Parameters:

- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive turns around the Led Ring. If 0, the animation continues endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – The service wait for the animation to finish or not to answer. If iterations is 0, the service answers immediately.
- **callback** (*function*) – Callback invoked on successful execution.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:

None (https://docs.python.org/3/library/constants.html#None)

snake(*color*, *period=0*, *iterations=0*, *wait=False*, *callback=None*, *timeout=None*)

A small coloured snake (certainly a python :D) runs around the LED ring.

Examples:

```
# Synchronous use
led_ring.snake([15, 50, 255]) # Non-blocking
led_ring.snake([15, 50, 255], 1, 100, True) # Blocking

# Asynchronous use
def led_ring_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Failed")
    else:
        print("Completed with success")

led_ring.snake([15, 50, 255], period=2, iterations=20, wait=True, callback=calibration_callback)
```

Parameters:

- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default duration will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive turns around the Led Ring. If 0, the animation continues endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – The service wait for the animation to finish or not to answer. If iterations is 0, the service answers immediately.
- **callback** (*function*) – Callback invoked on successful execution.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:

None (https://docs.python.org/3/library/constants.html#None)

rainbow(*period=0*, *iterations=0*, *wait=False*, *callback=None*, *timeout=None*)

Draw rainbow that fades across all LEDs at once.

Examples:

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
# Synchronous use
led_ring.rainbow()  # Non-blocking
led_ring.rainbow(5, 2, True)  # Blocking
led_ring.rainbow(wait=True)  # Blocking

# Asynchronous use
def led_ring_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Failed")
    else:
        print("Completed with success")

led_ring.rainbow(period=2, iterations=20, wait=True, callback=calibration_callback)
```

Parameters:

- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive rainbows. If 0, the animation continues endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – The service wait for the animation to finish or not to answer. If iterations is 0, the service answers immediately.
- **callback** (*function*) – Callback invoked on successful execution.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

rainbow_cycle(*period=0, iterations=0, wait=False, callback=None, timeout=None*)

Draw rainbow that uniformly distributes itself across all LEDs.

Examples:

```
# Synchronous use
led_ring.rainbow_cycle()
led_ring.rainbow_cycle(5, 2, True)
led_ring.rainbow_cycle(wait=True)

# Asynchronous use
def led_ring_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Failed")
    else:
        print("Completed with success")

led_ring.rainbow_cycle(period=2, iterations=20, wait=True, callback=calibration_callback)
```

Parameters:

- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive rainbow cycles. If 0, the animation continues endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – The service wait for the animation to finish or not to answer. If iterations is 0, the service answers immediately.
- **callback** (*function*) – Callback invoked on successful execution.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

rainbow_chase(*period=0, iterations=0, wait=False, callback=None, timeout=None*)

Rainbow chase animation, like the led_ring_chase method.

Examples:

```
# Synchronous use
led_ring.rainbow_chase()
led_ring.rainbow_chase(5, 2, True)
led_ring.rainbow_chase(wait=True)

# Asynchronous use
def led_ring_callback(result):
    if result["status"] < RobotErrors.SUCCESS.value:
        print("Failed")
    else:
        print("Completed with success")

led_ring.rainbow_chase(period=2, iterations=20, wait=True, callback=calibration_callback)
```

Parameters:

- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive rainbow cycles. If 0, the animation continues endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – The service wait for the animation to finish or not to answer. If iterations is 0, the service answers immediately.
- **callback** (*function*) – Callback invoked on successful execution.
- **timeout** (*float* (https://docs.python.org/3/library/functions.html#float)) – Timeout for the operation, in seconds. Only used if blocking.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

custom(*led_colors*)

Sends a colour command to all LEDs of the LED ring. The function expects a list of colours for the 30 LEDs of the robot.

Example:

```
led_list = [[i / 30. * 255 , 0, 255 - i / 30.] for i in range(30)]
led_ring.custom(led_list)

run_flag = True

def french_flag_moving():
    colors = []
    colors += [[255, 255, 255] for _ in range(2)]
    colors += [[0, 0, 255] for _ in range(11)]
    colors += [[255, 255, 255] for _ in range(4)]
    colors += [[255, 0, 0] for _ in range(11)]
    colors += [[255, 255, 255] for _ in range(2)]

    rate = 10
    while run_flag:
        for i in range(len(colors)):
            led_ring.custom(colors[i:] + colors[:i])
            time.sleep(1/rate)
            if not run_flag:
                return

french_flag_moving()
```

Parameters:
led_colors (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – List of size 30 of led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

set_led_color(*led_id*, *color*)

Lights up an LED in one colour. RGB colour between 0 and 255.

Example:

```
robot.set_led_color(5, [15, 50, 255])
```

Parameters:
- **led_id** (*int* (https://docs.python.org/3/library/functions.html#int)) – Id of the led: between 0 and 29
- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Led Ring - Niryo Topics

The use of these functions is explained in the NiryoTopics (index.html#niryotopic), section. They allow the acquisition of data in real time by callbacks or by direct call.

Led Ring's Niryo Topics

Name	Function	Return type
/status	status	LedRingStatusObject

Led Ring - Enums

List of enums:

- **AnimationMode**
- **LedMode**

class **AnimationMode**(*value*)

Enumeration of animations available for the user to control the Led Ring

NONE= *-1*

UNKNOWN= *0*

SOLID= *1*

FLASHING= *2*

ALTERNATE= *3*

CHASE= *4*

COLOR_WIPE= *5*

RAINBOW= *6*

RAINBOW_CYLE= *7*

RAINBOW_CHASE= *8*

GO_UP= *9*

GO_UP_AND_DOWN= *10*

BREATH= *11*

SNAKE= *12*

CUSTOM= *13*

class **LedMode**(*value*)

Enumeration of available Led Mode

NONE= *-1*

UNKNOWN= *0*

ROBOT_STATUS= *1*

USER= *2*

Led Ring - Objects

class **LedRingStatusObject**

Object used to store Led Ring status

Sound

This file presents the different functions, enums and topics available with the Sound API

Sound - Command functions

This section reference all existing functions to control the Sound interface of the Ned2. All functions are accessible via an instance of the class NiryoRobot (index.html#niryorobot)

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
robot = NiryoRobot(<robot_ip_address>)

robot.sound.play('connected.wav')
...
```

List of functions:

- Sound - Play
- Sound - Volume
- Sound - Manage

class **Sound**(*client*)

Sound robot functions

Example:

```
ros_instance = NiryoRos("127.0.0.1") # Hotspot
sound_interface = Sound(ros_instance)
```

Parameters:
client (*NiryoRos* (index.html#pyniryo2.niryo_ros.NiryoRos)) – Niryo ROS client

Sound - Play

Sound.play(*sound_name, wait_end=False, start_time_sec=0, end_time_sec=0*)

Play a sound that as already been imported by the user on the robot.

Example:

```
# If you know that the sound test_sound.wav is already imported on the robot
sound.play_sound_user("test_sound.wav")

# If you want to play the first sound of the ones that are already on the robot without knowing its name
sound_name = sound.get_sounds()[0]
sound_duration = sound.play(sound_name)

# Waits until the sound has been fully played
sound_duration = sound.play(sound_name, wait_end=True)

#  Doesn't wait until the sound has been fully played
sound_duration = sound.play(sound_name, wait_end=False)

# Plays sound from 1.1 seconds from start to 4.3 seconds from start
sound_duration = sound.play(sound_name, start_time_sec=1.1, end_time_sec=4.3)
```

Param:
sound_name: Name of the sound that will be played

Parameters:
- **wait_end** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – wait for the end of the sound before exiting the function
- **start_time_sec** (*float* (https://docs.python.org/3/library/functions.html#float)) – start the sound from this value in seconds
- **end_time_sec** (*float* (https://docs.python.org/3/library/functions.html#float)) – end the sound at this value in seconds

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Sound.stop()

Stop a sound being played. It will get automatically the name of the sound being played and stop it.

Example:

```
self.sound.stop()
```

Return type:
None (https://docs.python.org/3/library/constants.html#None)

property **Sound.state**

Returns the sound state client which can be used synchronously or asynchronously to obtain the current played sound.

Examples:

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
# Get last value
sound.state()
sound.state.value

# Subscribe a callback
def sound_callback(sound_name):
    print sound_name

sound.state.subscribe(sound_callback)
sound.state.unsubscribe()
```

Returns:
sound state topic instance

Return type:
NiryoTopic (index.html#pyniryo2.niryo_topic.NiryoTopic)

Sound.say(*text*, *language=<Language.ENGLISH: 0>*)

Use gtts (Google Text To Speech) to interpret a string as sound

Languages available are:

```
- English: Language. ENGLISH
- French: Language.FRENCH
- Spanish: Language.SPANISH
- Mandarin: Language.MANDARIN
- Portuguese: Language.PORTUGUESE
```

Example

```
robot.say("Hello", Language.ENGLISH)
robot.say("Bonjour", Language.FRENCH)
robot.say("Hola", Language.SPANISH)
```

Parameters:
- **text** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – Text that needs to be spoken < 100 char
- **language** (*Language* (index.html#pyniryo2.sound.enums.Language)) – language of the text

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Sound - Volume

property **Sound.volume**

Returns the volume state client which can be used synchronously or asynchronously to obtain the current volume.

Examples:

```
# Get last value
sound.volume()
sound.volume.value

# Subscribe a callback
def volume_callback(value):
    print value

sound.volume.subscribe(volume_callback)
sound.volume.unsubscribe()
```

Returns:
volume topic instance

Return type:
NiryoTopic (index.html#pyniryo2.niryo_topic.NiryoTopic)

Sound.get_volume()

Returns the volume of the robot. The sound can be set between 0 (sound off) and 100 (sound max)

Examples:

```
# Get the volume of the sound
sound.get_volume()
```

Returns:
int8 corresponding to the volume (0: sound off, 100: sound max)

Return type:
int8

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Sound.set_volume(*sound_volume*)

Set the volume of the robot. You can set it between 0 and 100 (0: sound off and 100: sound max). If you put less than 0, the volume will be set to 0. If you put more than 100, the volume will be set to 100.

Example:

```
# Set the volume to 25
self.sound.set_volume(25)
self.sound.play_sound_user("test_sound.wav")
```

Parameters:
sound_volume (*int8*) – Between O and 100 (0 sound off and 100 sound maximum)

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Sound - Manage

property **Sound.sounds**

Returns the list of available sounds in the robot

Examples:

```
sounds_list = sound.sounds
```

Returns:
Returns the list of available sounds in the robot

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[str (https://docs.python.org/3/library/stdtypes.html#str)]

Sound.get_sounds()

Returns the list of available sounds in the robot

Examples:

```
sounds_list = sound.get_sounds()
```

Returns:
Returns the list of available sounds in the robot

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[str (https://docs.python.org/3/library/stdtypes.html#str)]

Sound.save(*sound_name*, *sound_path*)

Import a sound on the RaspberryPi of the robot. To do that, you will need the encoded data from a wav or mp3 sound. It is preferable to put the encoded data from the sound on a text file and directly read it from this file. You also need to give the name of the sound you want to import.

Example:

```
sound_name = "test_import_sound.wav"
sound_path = "/home/niryo/test_sound.wav"

ros_instance = pyniryo2.NiryoRos("10.10.10.10")
sound = pyniryo2.Sound(ros_instance)
sound.save(sound_name, sound_path)
sound.play(sound_name)
```

Parameters:

- **sound_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – For example, test.wav. Il will be the name of the sound in the robot
- **sound_path** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – absolute path to the sound file

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Sound.delete(*sound_name*)

Delete a sound imported on the robot

Example:

```
self.sound.delete("test_sound.wav")
```

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Parameters:
sound_name (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – For example, test.wav

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Sound.get_sound_duration(*sound_name*)

Get the duration of a sound in seconds

Examples:

```
sound_name = sound.get_sounds()[0]
sound_duration = sound.get_sound_duration(sound_name)
sound_duration = sound.get_sound_duration('test_sound.mp3')
```

Returns:
Returns the duration of a sound in seconds

Return type:
float (https://docs.python.org/3/library/functions.html#float)

Sound - Niryo Topics

The use of these functions is explained in the NiryoTopics (index.html#niryotopic), section. They allow the acquisition of data in real time by callbacks or by direct call.

Sound's Niryo Topics

Name	Function	Return type
/niryo_robot_sound/sound	**state**	**str** (https://docs.python.org/3/library/stdtypes.html#str)
/niryo_robot_sound/sound_database	**sounds**	**dict** (https://docs.python.org/3/library/stdtypes.html#dict)
/niryo_robot_sound/volume	**volume**	**int** (https://docs.python.org/3/library/functions.html#int)

Sound - Enums

List of enums:

- **ManageSound**
- **Language**

class **ManageSound**(*value*)

Enumeration of the actions of sound database management

ADD= *1*

DELETE= *2*

class **Language**(*value*)

Enumeration of the Text To Speech languages

ENGLISH= *0*

FRENCH= *1*

SPANISH= *3*

MANDARIN= *4*

PORTUGUESE= *5*

Indices and tables

- Index (genindex.html)
- Module Index (py-modindex.html)
- Search Page (search.html)

Suggest a modification Download as PDF

2022, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS